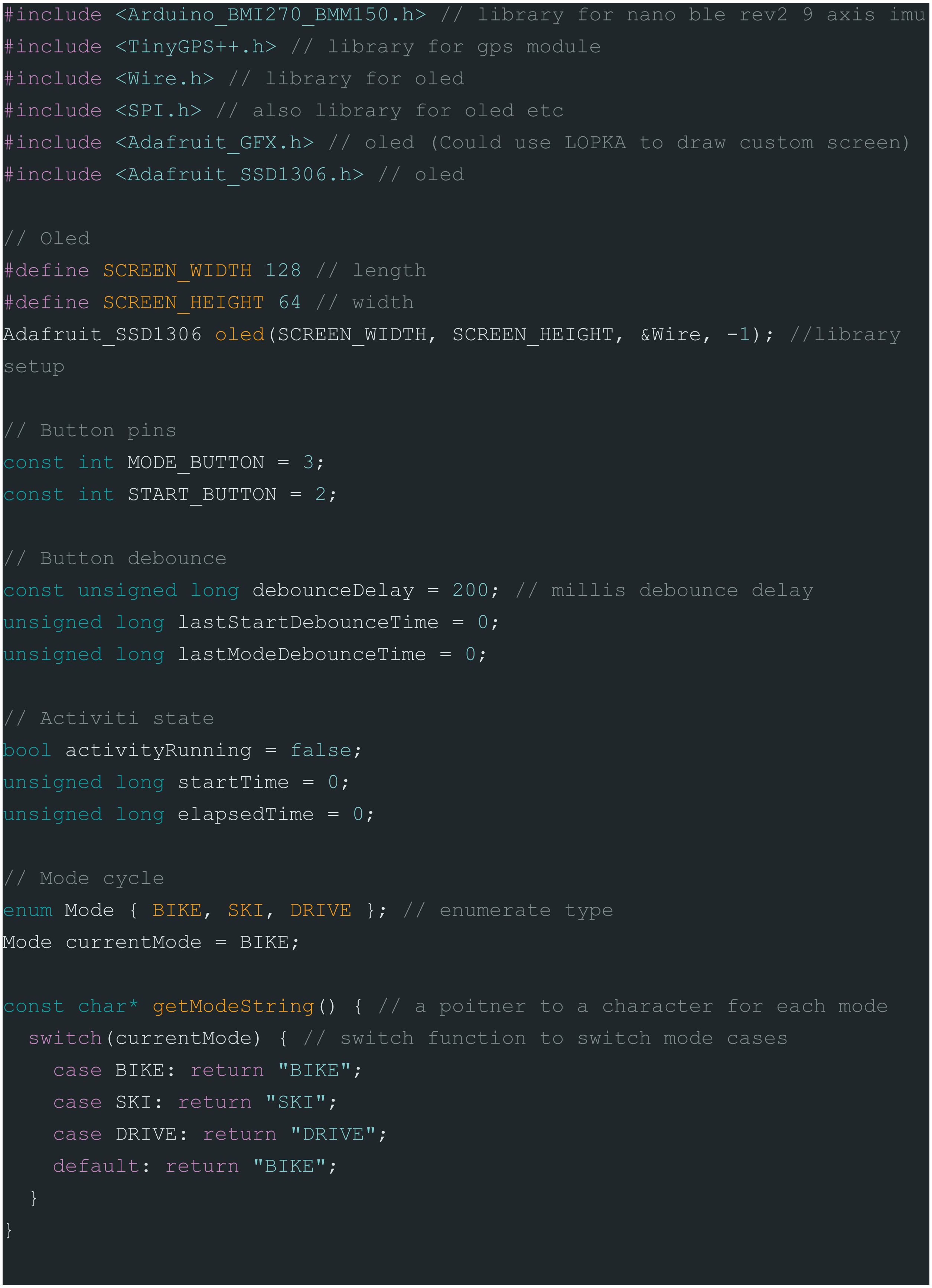
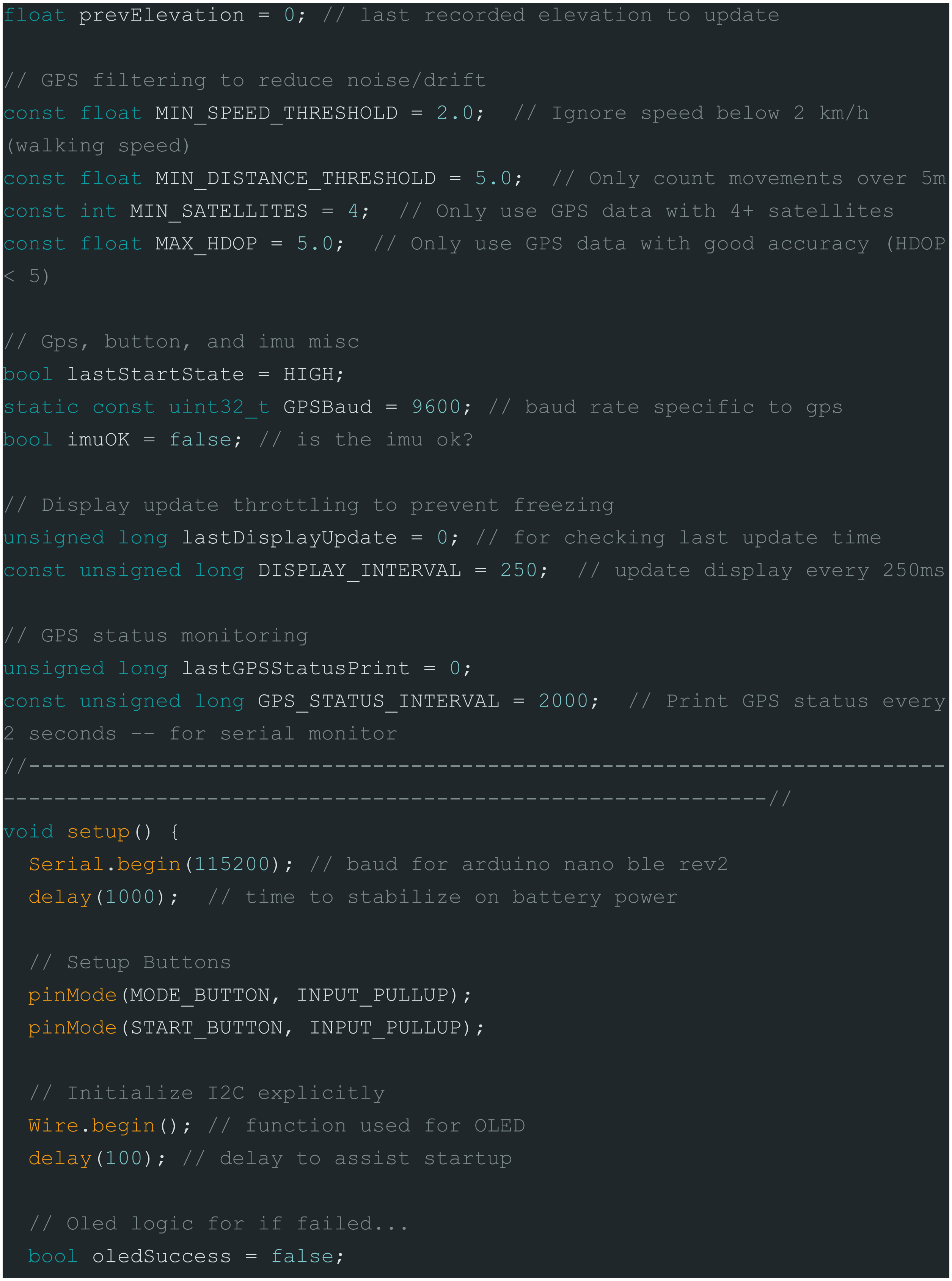


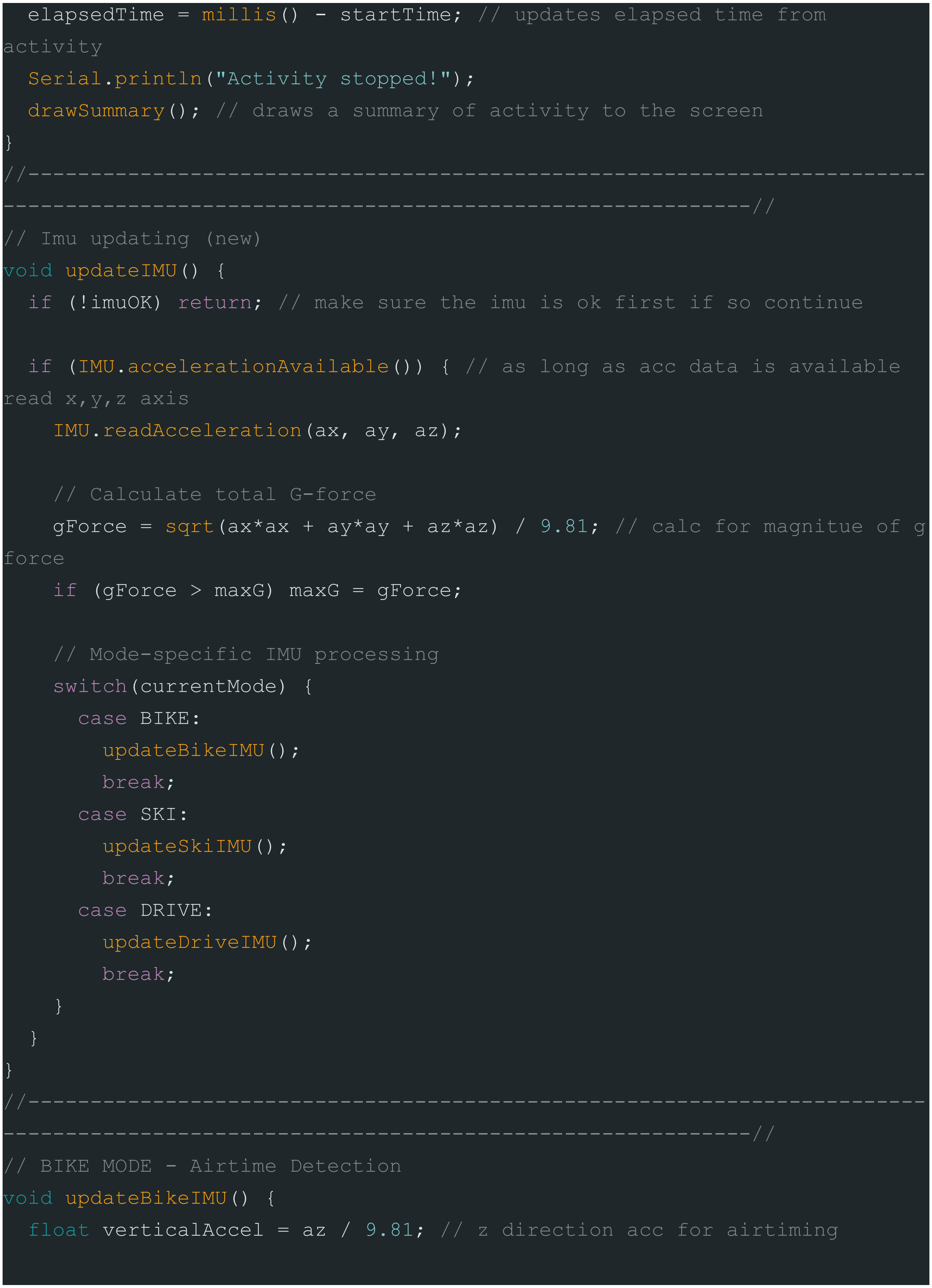
```
#include <Arduino_BMI270_BMM150.h> // library for nano ble rev2 9 axis imu
#include <TinyGPS++.h> // library for gps module
#include <Wire.h> // library for oled
#include <SPI.h> // also library for oled etc
#include <Adafruit_GFX.h> // oled (Could use LOPKA to draw custom screen)
#include <Adafruit_SSD1306.h> // oled

// Oled
#define SCREEN_WIDTH 128 // length
#define SCREEN_HEIGHT 64 // width
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1); //library setup

// Button pins
const int MODE_BUTTON = 3;
const int START_BUTTON = 2;

// Button debounce
const unsigned long debounceDelay = 200; // millis debounce delay
unsigned long lastStartDebounceTime = 0;
unsigned long lastModeDebounceTime = 0;

// Activiti state
bool activityRunning = false;
unsigned long startTime = 0;
unsigned long elapsedTime = 0;

// Mode cycle
enum Mode { BIKE, SKI, DRIVE }; // enumerate type
Mode currentMode = BIKE;

const char* getModeString() { // a poitner to a character for each mode
  switch(currentMode) { // switch function to switch mode cases
    case BIKE: return "BIKE";
    case SKI: return "SKI";
    case DRIVE: return "DRIVE";
    default: return "BIKE";
  }
}
```

```
// Variables for IMU
float ax, ay, az; // acceleration in x,y,z directions
float gForce = 0; // gforce variable
float maxG = 0; // used to update max g

// BIKE MODE - Airtime Detection
const float BIKE_AIRTIME_THRESHOLD = -1.5; // used for managing airtime
detection (could be changed based on ridng)
const unsigned long BIKE_MIN_AIRTIME_MS = 200; // minimum time to record
airtime (could be changed based on ridng)

// SKI MODE - Airtime Detection
const float SKI_AIRTIME_THRESHOLD = -2.5;  // longer and smoother jumps
for skiing
const unsigned long SKI_MIN_AIRTIME_MS = 250;  // larger time because ski
jumps are usually bigger

// DRIVE MODE - Acceleration tracking
float maxAccelG = 0;  // Maximum acceleration G-force
float zeroToSixtyTime = 0;  // Best 0-60 km/h time in seconds - should
change to 102 kph (accurate to 0-60 in mph)
bool timingZeroToSixty = false; // current 0-60 recording?
unsigned long zeroToSixtyStart = 0; // start for 0-60 recording
const float ACCEL_START_THRESHOLD = 5.0;  // Start timing above 5 km/h
const float ACCEL_END_THRESHOLD = 60.0;  // End timing at 60 km/h - need
to change to 100 kph

// Shared airtime variables
unsigned long airtimeStart = 0;
unsigned long totalAirtime = 0;
bool inAir = false; // are we in the air?

// GPS Variables
TinyGPSPlus gps;
double prevLat = 0, prevLon = 0; // start/update variable for lat and lon
double totalDistance = 0;
float currentSpeed = 0;
float maxSpeed = 0;
float elevation = 0; // based on elevation gain (sometimes works for
decents)- need to adjust
```



```
float prevElevation = 0; // last recorded elevation to update

// GPS filtering to reduce noise/drift
const float MIN_SPEED_THRESHOLD = 2.0;  // Ignore speed below 2 km/h
(walking speed)
const float MIN_DISTANCE_THRESHOLD = 5.0;  // Only count movements over 5m
const int MIN_SATELLITES = 4;  // Only use GPS data with 4+ satellites
const float MAX_HDOP = 5.0;  // Only use GPS data with good accuracy (HDOP
< 5)

// Gps, button, and imu misc
bool lastStartState = HIGH;
static const uint32_t GPSBaud = 9600; // baud rate specific to gps
bool imuOK = false; // is the imu ok?

// Display update throttling to prevent freezing
unsigned long lastDisplayUpdate = 0; // for checking last update time
const unsigned long DISPLAY_INTERVAL = 250;  // update display every 250ms

// GPS status monitoring
unsigned long lastGPSStatusPrint = 0;
const unsigned long GPS_STATUS_INTERVAL = 2000;  // Print GPS status every
2 seconds -- for serial monitor
//------------------------------------------------------------------------
------------------------------------------------------------//
void setup() {
  Serial.begin(115200); // baud for arduino nano ble rev2
  delay(1000);  // time to stabilize on battery power

  // Setup Buttons
  pinMode(MODE_BUTTON, INPUT_PULLUP);
  pinMode(START_BUTTON, INPUT_PULLUP);

  // Initialize I2C explicitly
  Wire.begin(); // function used for OLED
  delay(100); // delay to assist startup

  // Oled logic for if failed...
  bool oledSuccess = false;
  for (int attempt = 0; attempt < 3; attempt++) { //try for "4" attempts
(all printed to serial monitor)
    if (oled.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
      oledSuccess = true;
      Serial.println("OLED initialized");
      break;
    }
    Serial.print("OLED init attempt ");
    Serial.print(attempt + 1);
    Serial.println(" failed, retrying..."); // if it fails add to attempt
and retry
    delay(500);
  }

  if (!oledSuccess) {
    Serial.println("OLED init failed after 3 trys"); // if no success
print fail to serial monitor
  } else {
    oled.clearDisplay();
    oled.display();
  }

  // IMU
  imuOK = IMU.begin();
  Serial.println(imuOK ? "IMU OK" : "IMU FAIL"); // check if imu is ok
(bool) if yes print ok if not print fail to serial monitor
  if (!imuOK) {
    Serial.println("Failed to initialize IMU!"); // if not okay print init
failed to serial monitor.
  }

  // GPS Serial1
  Serial1.begin(GPSBaud); // begin serial1 for GPS

  // Initial screen
  if (oledSuccess) {
    drawScreen(); // use draw screen function to load default screen to
OLED
  }
}
```

```
//------------------------------------------------------------------------------------------------------------------------------------//
void loop() {
  checkStartButton(); // continuously check start button function to see if activity started/stopped
  checkModeButton();  // continuously check mode to see if mode changes (cant change mode when activity currently running)

  // GPS data
  while (Serial1.available() > 0) { // always write gps data to serial1 as long as its available (basically always)
    gps.encode(Serial1.read());
  }

  if (activityRunning) { // if an activity is running run these functions (IMU function uses each mode case within)
    updateIMU();
    updateGPSData();
    updateTimer();

    // Throttle display updates to prevent blocking
    unsigned long currentTime = millis();
    if (currentTime - lastDisplayUpdate >= DISPLAY_INTERVAL) {
      lastDisplayUpdate = currentTime;
      drawScreen();
    }
  }

  // GPS Status Monitor (non-blocking, runs regardless of activity state)
  printGPSStatus();
}
//------------------------------------------------------------------------------------------------------------------------------------//
// Button stuff
void checkStartButton() { // check activity button function
  bool pressed = (digitalRead(START_BUTTON) == LOW); //created pressed bool within for simplicity to read start button
  unsigned long currentTime = millis(); // millis timing setup
  if (pressed && lastStartState == HIGH && (currentTime - 
lastStartDebounceTime) > debounceDelay) { // use millis delay and 
start/stop activity depending on what the board reads
    lastStartDebounceTime = currentTime;
    if (!activityRunning) {
      startActivity();
    } else {
      stopActivity();
    }
  }

  lastStartState = pressed; // resset start state
}
//-------------------------------------------------------------------------
------------------------------------------------------------//
void checkModeButton() { // mode button check function
  bool modePressed = (digitalRead(MODE_BUTTON) == LOW); // another bool 
for if button mode button is pressed within function
  static bool lastModeState = HIGH; // static bool for last mode state 
(default high - true)
  unsigned long currentTime = millis(); // millis timing

  // Only allow mode change when activity is not running
  if (modePressed && lastModeState == HIGH && !activityRunning &&
      (currentTime - lastModeDebounceTime) > debounceDelay) {
    lastModeDebounceTime = currentTime;

    // Cycle through modes
    switch(currentMode) {
      case BIKE:
        currentMode = SKI;
        break;
      case SKI:
        currentMode = DRIVE;
        break;
      case DRIVE:
        currentMode = BIKE;
        break;
    }
    Serial.print("Mode changed to: "); //print to serial monitor
    Serial.println(getModeString());
    drawScreen();  // Update display immediately
  }

  lastModeState = modePressed; //reset last mode state
}
//---------------------------------------------------------------------------
------------------------------------------------------------//
void startActivity() { // starts activity (used in check start button
function)
  activityRunning = true;
  startTime = millis(); // keep track of time

  // Reset all tracked data after summary
  totalAirtime = 0;
  maxG = 0;
  totalDistance = 0;
  maxSpeed = 0;
  elevation = 0;
  prevElevation = 0;
  prevLat = 0;
  prevLon = 0;
  inAir = false;
  airtimeStart = 0;

  // Drive mode specific resets
  maxAccelG = 0;
  zeroToSixtyTime = 0;
  timingZeroToSixty = false;

  Serial.print("Activity started in ");
  Serial.print(getModeString()); // function to get mode case
  Serial.println(" mode!"); // serial monitor
}
//---------------------------------------------------------------------------
-----------------------------------------------------------//
void stopActivity() { // stops activity (used in check start button
function)
  activityRunning = false; // sets running activity to false
```



```
  elapsedTime = millis() - startTime; // updates elapsed time from
activity
  Serial.println("Activity stopped!");
  drawSummary(); // draws a summary of activity to the screen
}
//------------------------------------------------------------------------
------------------------------------------------------------//
// Imu updating (new)
void updateIMU() {
  if (!imuOK) return; // make sure the imu is ok first if so continue

  if (IMU.accelerationAvailable()) { // as long as acc data is available
read x,y,z axis
    IMU.readAcceleration(ax, ay, az);

    // Calculate total G-force
    gForce = sqrt(ax*ax + ay*ay + az*az) / 9.81; // calc for magnitue of g
force
    if (gForce > maxG) maxG = gForce;

    // Mode-specific IMU processing
    switch(currentMode) {
      case BIKE:
        updateBikeIMU();
        break;
      case SKI:
        updateSkiIMU();
        break;
      case DRIVE:
        updateDriveIMU();
        break;
    }
  }
}
//------------------------------------------------------------------------
------------------------------------------------------------//
// BIKE MODE - Airtime Detection
void updateBikeIMU() {
  float verticalAccel = az / 9.81; // z direction acc for airtiming
  if (verticalAccel < BIKE_AIRTIME_THRESHOLD && !inAir) { // function that starts airtime recording once detected
    airtimeStart = millis();
    inAir = true; // sets in air variable to high/true
    Serial.println("Airtime started (BIKE)"); // serial monitor only
  }
  else if (verticalAccel > 0.5 && inAir) { // sets air time
    unsigned long airDuration = millis() - airtimeStart;

    if (airDuration >= BIKE_MIN_AIRTIME_MS) {
      totalAirtime += airDuration; // adds to total (keeps a running summary)
      Serial.print("Airtime logged: "); // serial monitor only
      Serial.print(airDuration);
      Serial.println(" ms");
    }

    inAir = false; // resets air time
    airtimeStart = 0;
  }
}
//----------------------------------------------------------------------------------------------------------------------------------//
// SKI MODE - Airtime Detection (exact same as bike mode but uses different thresholds for less sensitivity)
void updateSkiIMU() {
  float verticalAccel = az / 9.81;

  if (verticalAccel < SKI_AIRTIME_THRESHOLD && !inAir) {
    airtimeStart = millis();
    inAir = true;
    Serial.println("Airtime started (SKI)");
  }
  else if (verticalAccel > 0.5 && inAir) {
    unsigned long airDuration = millis() - airtimeStart;

    if (airDuration >= SKI_MIN_AIRTIME_MS) {
      totalAirtime += airDuration;
      Serial.print("Airtime logged: ");
      Serial.print(airDuration);
      Serial.println(" ms");
    }

    inAir = false;
    airtimeStart = 0;
  }
}
//--------------------------------------------------------------------------
-------------------------------------------------------------//
// DRIVE MODE - Acceleration tracking
void updateDriveIMU() {
  // Track maximum acceleration G-force (forward acceleration)
  float forwardAccel = ax / 9.81;  // ax is forward/backward
  if (abs(forwardAccel) > maxAccelG) {
    maxAccelG = abs(forwardAccel);
  }

  // 0-60 km/h timing - need to change to 102 kph
  if (gps.speed.isValid()) {
    float speed = currentSpeed;

    // Start timing when crossing 5 km/h threshold
    if (!timingZeroToSixty && speed > ACCEL_START_THRESHOLD && speed <
ACCEL_END_THRESHOLD) {
      timingZeroToSixty = true; // now true (timing 0-60)
      zeroToSixtyStart = millis(); // start timer
      Serial.println("0-60 timing started");
    }

    // Stop timing when crossing 60 km/h - change to 102 kph
    if (timingZeroToSixty && speed >= ACCEL_END_THRESHOLD) {
      float timeSeconds = (millis() - zeroToSixtyStart) / 1000.0;
//converts ms to s for recorded 0-60 (change to 102)

      // Records best/or only 0-60 (change to 102)
      if (zeroToSixtyTime == 0 || timeSeconds < zeroToSixtyTime) {
        zeroToSixtyTime = timeSeconds;
        Serial.print("New 0-60 time: ");
        Serial.print(zeroToSixtyTime, 2);
        Serial.println(" seconds");
      }

      timingZeroToSixty = false; // reset (not recording 0-60 anymore)
    }

    // Reset timing if speed drops back below threshold
    if (timingZeroToSixty && speed < ACCEL_START_THRESHOLD) {
      timingZeroToSixty = false;
      Serial.println("0-60 timing cancelled (speed dropped)");
    }
  }
}
//------------------------------------------------------------------------
------------------------------------------------------------//
// Gps updating (new)
void updateGPSData() {
  // Only update if have new valid location data
  if (gps.location.isUpdated() && gps.location.isValid()) {

    // Check GPS satellites available before tracking data (needs 4 to
start)
    bool goodGPSFix = true;

    // Require minimum satellites
    if (gps.satellites.isValid() && gps.satellites.value() <
MIN_SATELLITES) { // 4 sats to start
      goodGPSFix = false;
    }

    // Require good accuracy (HDOP)
    if (gps.hdop.isValid() && gps.hdop.hdop() > MAX_HDOP) {
      goodGPSFix = false;
    }

    if (!goodGPSFix) {
      return;  // Skip GPS update if quality is poor or not enough
satellites
    }

    double lat = gps.location.lat();
    double lon = gps.location.lng();

    // Speed - with threshold to ignore drift
    if (gps.speed.isValid()) {
      float rawSpeed = gps.speed.kmph();

      // Only update speed if above threshold
      if (rawSpeed >= MIN_SPEED_THRESHOLD) { // update speed threshold for more/less sensitivity if above threshold update currspeed
        currentSpeed = rawSpeed; //
        if (currentSpeed > maxSpeed) maxSpeed = currentSpeed; // set max speed if curr speed is the max recorded value so far
      } else {
        currentSpeed = 0;  // Show 0 when basically stationary
      }
    }

    // Distance - with minimum movement threshold
    if (prevLat != 0 && prevLon != 0) {
      float distanceMoved = gps.distanceBetween(prevLat, prevLon, lat, lon);

      // Only add distance if movement is significant
      if (distanceMoved >= MIN_DISTANCE_THRESHOLD) { //need to update for mtb climbs/slow walking for more sensitivity
        totalDistance += distanceMoved; // keep running total of distance traveled
      }
    }
    prevLat = lat; // update lat
    prevLon = lon; // update lon

    // Elevation
    if (gps.altitude.isValid()) { // if gps alt is ok
      float alt = gps.altitude.meters(); // get curr altitude in m
      if (alt != 0 && prevElevation != 0) {
        float gain = alt - prevElevation; // record elevation gain - need to add a feature to get more accurate decent too
        if (gain > 0) elevation += gain;
      }
      prevElevation = alt; // reset previous elevation
    }
  }
}
//--------------------------------------------------------------------------
--------------------------------------------------------------//
// Timer
void updateTimer() {
  elapsedTime = millis() - startTime; // update main timer
}
//--------------------------------------------------------------------------
--------------------------------------------------------------//
// Gps status monitoring for serial monitor
void printGPSStatus() {
  unsigned long currentTime = millis();

  // Only print every GPS_STATUS_INTERVAL milliseconds
  if (currentTime - lastGPSStatusPrint >= GPS_STATUS_INTERVAL) { // need
so it doesn't freeze
    lastGPSStatusPrint = currentTime; // update last time data was printed

    Serial.println(" GPS STATUS "); // print status of accessible gps
features to serial monitor below

    // Satellites
    Serial.print("Satellites: ");
    if (gps.satellites.isValid()) {
      Serial.println(gps.satellites.value());
    } else {
      Serial.println("NO DATA");
    }

    // Location
    Serial.print("Location: ");
    if (gps.location.isValid()) {
      Serial.print(gps.location.lat(), 6);
      Serial.print(", ");
      Serial.print(gps.location.lng(), 6);
      Serial.print(" (Age: ");
      Serial.print(gps.location.age());
      Serial.println(" ms)");
    } else {
      Serial.println("INVALID");
    }

    // HDOP (accuracy indicator - lower is better, <5 is good)
    Serial.print("HDOP: ");
    if (gps.hdop.isValid()) {
      Serial.println(gps.hdop.hdop());
    } else {
      Serial.println("NO DATA");
    }

    // Speed
    Serial.print("Speed: ");
    if (gps.speed.isValid()) {
      Serial.print(gps.speed.kmph(), 1);
      Serial.println(" km/h");
    } else {
      Serial.println("NO DATA");
    }

    // Altitude
    Serial.print("Altitude: ");
    if (gps.altitude.isValid()) {
      Serial.print(gps.altitude.meters(), 1);
      Serial.println(" m");
    } else {
      Serial.println("NO DATA");
    }

    // Characters processed vs failed
    Serial.print("Chars: ");
    Serial.print(gps.charsProcessed());
    Serial.print(" | Failed: ");
    Serial.println(gps.failedChecksum());

    Serial.println("                 ");
    Serial.println();
  }
```

```
}
//----------------------------------------------------------------------------
------------------------------------------------------------//
// Oled display setup
void drawScreen() { // main default screen function that uses other mode
specific screens to update oled
  oled.clearDisplay(); // first clear
  oled.setTextSize(1); // set font size
  oled.setTextColor(SSD1306_WHITE); // color

  // Mode-specific display
  switch(currentMode) {
    case BIKE:
      drawBikeScreen();
      break;
    case SKI:
      drawSkiScreen();
      break;
    case DRIVE:
      drawDriveScreen();
      break;
  }

  oled.display(); // update oled
}
//----------------------------------------------------------------------------
------------------------------------------------------------//
// BIKE MODE Display
void drawBikeScreen() {
  // Top section
  oled.setCursor(0, 0); // start in top left corner
  oled.print("MODE: BIKE"); // print mode

  oled.setCursor(80, 0); // move to top right corner (basically)
  oled.print("Sats: "); // print satellite count
  if (gps.satellites.isValid()) {
    oled.print(gps.satellites.value());
  } else {
    oled.print("--"); // if no sats available print blank line
  }
  // Time
  oled.setCursor(0, 18); // move down and left for time section
  oled.print("Time: ");
  oled.print(elapsedTime / 1000);
  oled.print(" s");

  // Speed
  oled.setCursor(0, 28); // move down and print speed
  oled.print("Speed: ");
  if (gps.speed.isValid()) {
    oled.print(currentSpeed, 1);
  } else {
    oled.print("---"); // if no speed reading from gps print blank line
  }
  oled.print(" km/h");

  // Distance
  oled.setCursor(0, 38); // move down and print distance
  oled.print("Dist: ");
  oled.print(totalDistance / 1000.0, 2);
  oled.print(" km");

  // G-force
  oled.setCursor(0, 48); // move down and print g force
  oled.print("G: ");
  oled.print(maxG, 2);

  // Airtime
  oled.setCursor(0, 56); // down close to bottom and print airtime sum
  oled.print("Air: ");
  oled.print(totalAirtime / 1000.0, 2);
  oled.print(" s");
}
//----------------------------------------------------------------------------
---------------------------------------------------------------//
// SKI MODE Display
void drawSkiScreen() { // basically the exact same as bike mode but
replace current speed for current elevation sum
  // Top section
```

```
oled.setCursor(0, 0);
oled.print("MODE: SKI");

oled.setCursor(80, 0);
oled.print("Sats: ");
if (gps.satellites.isValid()) {
  oled.print(gps.satellites.value());
} else {
  oled.print("--");
}

// Time
oled.setCursor(0, 18);
oled.print("Time: ");
oled.print(elapsedTime / 1000);
oled.print(" s");

// Speed
oled.setCursor(0, 28);
oled.print("Speed: ");
if (gps.speed.isValid()) {
  oled.print(currentSpeed, 1);
} else {
  oled.print("---");
}
oled.print(" km/h");

// Distance
oled.setCursor(0, 38);
oled.print("Dist: ");
oled.print(totalDistance / 1000.0, 2);
oled.print(" km");

// Elevation
oled.setCursor(0, 48);
oled.print("Elev: ");
oled.print(elevation, 1);
oled.print(" m");

// Airtime
```

```
  oled.setCursor(0, 56);
  oled.print("Air: ");
  oled.print(totalAirtime / 1000.0, 2);
  oled.print(" s");
}
//--------------------------------------------------------------------------
-------------------------------------------------------------//
// DRIVE MODE Display
void drawDriveScreen() { // function to draw drive specific screen (more
different to bike and ski mode)
  // Top section
  oled.setCursor(0, 0); // also start at top left and print mode
  oled.print("MODE: DRIVE");

  oled.setCursor(80, 0); // also print satellite count in top right
  oled.print("Sats: ");
  if (gps.satellites.isValid()) {
    oled.print(gps.satellites.value());
  } else {
    oled.print("--"); // if no valid satellite count reading print blank
line
  }

  // Current Speed
  oled.setCursor(0, 18); // print current speed to screen
  oled.print("Speed: ");
  if (gps.speed.isValid()) {
    oled.print(currentSpeed, 1);
  } else {
    oled.print("---"); // if no valid speed reading print blank line
  }
  oled.print(" km/h"); // print kph

  // Top Speed
  oled.setCursor(0, 28); // print recorded top speed (below speed)
  oled.print("Top: ");
  oled.print(maxSpeed, 1);
  oled.print(" km/h");

  // Distance
  oled.setCursor(0, 38); // print distance sum (below Top Speed)
  oled.print("Dist: ");
  oled.print(totalDistance / 1000.0, 2);
  oled.print(" km");

  // Max Accel G
  oled.setCursor(0, 48); // print max acceleration G (below distance)
  oled.print("Max G: ");
  oled.print(maxAccelG, 2);

  // 0-60 time
  oled.setCursor(0, 56); // print best 0-60 (change to 102) (below max accel g)
  oled.print("0-60: ");
  if (zeroToSixtyTime > 0) {
    oled.print(zeroToSixtyTime, 2);
    oled.print(" s");
  } else {
    oled.print("---"); // if no reading or stays below threshold for complete activity print blank line
  }
}
//------------------------------------------------------------------------------------------------------------------------------------//
void drawSummary() { // main function to draw each mode summary to the screen after the activity has ended
  oled.clearDisplay(); // clear display
  oled.setTextSize(1); // set font size
  oled.setCursor(0, 0); // start in top left

  // Mode-specific summary
  switch(currentMode) { // write mode specific summary to the oled
    case BIKE:
      drawBikeSummary();
      break;
    case SKI:
      drawSkiSummary();
      break;
    case DRIVE:
      drawDriveSummary();
      break;
  }

  oled.display(); //reset/clear display
}
//----------------------------------------------------------------------------
---------------------------------------------------------------//
// BIKE MODE Summary
void drawBikeSummary() {
  oled.println("=== BIKE SUMMARY ==="); // top (header)
  oled.print("Time: "); // defaults to left side
  oled.print(elapsedTime / 1000); // prints total time in sec
  oled.println(" s");

  oled.print("Distance: "); // print total dist recorded - need to update
for slower climbs
  oled.print(totalDistance / 1000.0, 2); // converts units
  oled.println(" km");

  oled.print("Top speed: "); // prints highest speed recorded
  oled.print(maxSpeed, 1);
  oled.println(" km/h");

  oled.print("Max G: "); // print highest recorded g force (z axis)
  oled.print(maxG, 2);
  oled.println();

  oled.print("Airtime: "); // prints airtime if any is recorded
  oled.print(totalAirtime / 1000.0, 2);
  oled.println(" s");

  oled.print("Elevation: "); // prints elevation gain - need to adjust
descent
  oled.print(elevation, 1);
  oled.println(" m");
}
//----------------------------------------------------------------------------
---------------------------------------------------------------//
// SKI MODE Summary
void drawSkiSummary() {
  oled.println("=== SKI SUMMARY ==="); // header for ski mode
  oled.print("Time: "); // defaults to left side
  oled.print(elapsedTime / 1000); // prints total time in sec
  oled.println(" s");

  oled.print("Distance: "); // print total dist recorded
  oled.print(totalDistance / 1000.0, 2); // converts units
  oled.println(" km");

  oled.print("Top speed: "); // prints highest speed recorded
  oled.print(maxSpeed, 1);
  oled.println(" km/h");

  oled.print("Elevation: "); // prints elevation change
  oled.print(elevation, 1);
  oled.println(" m");

  oled.print("Max G: "); // prints highest recorded g force (less sensitive)
  oled.print(maxG, 2);
  oled.println();

  oled.print("Airtime: "); // prints airtime sum
  oled.print(totalAirtime / 1000.0, 2);
  oled.println(" s");
}
//------------------------------------------------------------------------------------------------------------------------------------//
// DRIVE MODE Summary
void drawDriveSummary() {
  oled.println("=== DRIVE SUMMARY ==="); // drive mode header
  oled.print("Time: "); // defaults to left - prints time
  oled.print(elapsedTime / 1000); // seconds
  oled.println(" s");

  oled.print("Distance: "); // prints total distance
  oled.print(totalDistance / 1000.0, 2); // converts units
  oled.println(" km");

  oled.print("Top speed: "); // prints highest speed recorded
  oled.print(maxSpeed, 1);
  oled.println(" km/h");

  oled.print("Max Accel: "); // prints max accel (uses different axis
compared to bike mode)
  oled.print(maxAccelG, 2);
  oled.println(" G");

  oled.print("0-60 km/h: "); // prints best 0-60 but need to update to 102
(for us americans)
  if (zeroToSixtyTime > 0) { // only prints time if one was recorded
    oled.print(zeroToSixtyTime, 2);
    oled.println(" s");
  } else {
    oled.println("N/A"); // if no recorded time just print N/A
  }

  oled.print("Max G: "); // prints max g force in specified direction/axis
  oled.print(maxG, 2);
  oled.println();
}
/*
APPENDIX SECTION
--------------------------------------------------------------------------
-------------------------------------------//
References:

[1] Arduino, "Arduino Nano 33 BLE Rev2," Arduino Documentation. [Online].
    Available: https://docs.arduino.cc/hardware/nano-33-ble-rev2/.
    [Accessed: Nov 15, 2025].

[2] M. Mikalsen, "TinyGPSPlus Library," GitHub. [Online].
    Available: https://github.com/mikalhart/TinyGPSPlus.
    [Accessed: Nov 20, 2025].

[3] Adafruit Industries, "Adafruit SSD1306 Library," GitHub. [Online].
    Available: https://github.com/adafruit/Adafruit_SSD1306.
    [Accessed: Nov 20, 2025].

[4] Adafruit Industries, "Adafruit GFX Library," GitHub. [Online].
```

```
    Available: https://github.com/adafruit/Adafruit-GFX-Library.
    [Accessed: Nov 20, 2025].

[5] Arduino, "Arduino_BMI270_BMM150 Library," Arduino Libraries. [Online].
    Available:
https://www.arduino.cc/reference/en/libraries/arduino_bmi270_bmm150/.
    [Accessed: Nov 20, 2025].

[6] LastMinuteEngineers, "In-Depth: Interface NEO-6M GPS Module with
Arduino,"
    Last Minute Engineers. [Online].
    Available:
https://lastminuteengineers.com/neo6m-gps-arduino-tutorial/.
    [Accessed: Nov 15, 2025].

[7] LastMinuteEngineers, "Interface OLED Graphic Display Module with
Arduino,"
    Last Minute Engineers. [Online].
    Available:
https://lastminuteengineers.com/oled-display-arduino-tutorial/.
    [Accessed: Nov 15, 2025].

[8] u-blox, "NEO-6 Series GPS Module Datasheet," u-blox AG, 2011.
[Online].
    Available: https://www.u-blox.com/en/product/neo-6-series

[9] HiLetgo, "TP4056 Type-C USB 5V 1A Lithium Battery Charger Module,"
     Product Documentation, Amazon. [Online].
     Available: https://www.amazon.com. [Accessed: Nov 21, 2025].

[10] Anthropic, "Claude (Claude Sonnet 4.5)," Nov-Dec. 2025. [Online].
    Available: https://claude.ai

*/
```