```
//Importing all the necessary libraries
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>
#include <Servo.h>
#include <Stepper.h>

// Using Adafruit_ADXL345_Unified library to create an object called 'accel'.  Then
assigning it an random ID (12345 in this case)
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);

//Pin to control DC motor speed
const int dcmotorPin = 3;

// 7-segment shift register pins
const int dataPin  = 8; //SER pin used to send bits into the shift register
const int latchPin = 11; //RCLK pin used to send bits from shift register to
storage register
const int clockPin = 12; //SRCLK pin creates a new slot in shift register (upto 8)
for SER to send its values into

// Servo pin
const int servoPin = 6;
Servo speedServo; //Creating a 'speedServo' object for this particular instance
using Servo.h library

// Joystick pins for x axis, y axis and button inputs
const int joyXPin = A0;
const int joyYPin = A1;
const int joyButtonPin = 7;

// Ultrasonic sensor pins
const int trigPin = 5;
const int echoPin = 4;

// Stepper motor
const int stepsPerRevolution = 2048;
Stepper myStepper(stepsPerRevolution, 13, 10, 9, 2); //Creating a 'myStepper'
object for this particular instance
const int joyCenter = 512; //represents the center position of analog output
const int joyDeadzone = 50; //a value of +- 50 around the center which will also be
considered as center to avoid noise

// 7-segment digit encodings as an array of bite values
const byte digitSegments[] = {
  0b00111111, // digit 0
  0b00000110, // digit 1
  0b01011011, // digit 2
  0b01001111, // digit 3
  0b01100110, // digit 4
```

```
  0b01101101, // digit 5
  0b01111101, // digit 6
  0b00000111, // digit 7
  0b01111111, // digit 8
  0b01101111  // digit 9
};

// Defining fall detection thresholds based on physical observations
const float lowerThreshold = 0.6 * 9.81;
const float upperThreshold = 1.5 * 9.81;

bool droneMode = false;// Drone mode being switched off by default
int lastButtonState = HIGH;// variable to store the previous reading obtained from
joystick button

int currentSpeed = 0; //variable to store speed in arbitary units
const float tiltThreshold = 0.1; //changes in acceleration values less than 0.1 g
will be disregarded as noise
const int speedStep = 5; //amount by which the speed will change

// setup loop to initialize sensors, actuators, and pin modes.
void setup() {
  Serial.begin(9600); //to observe serial ouptput

  //simple loop to test the accelerometer connections
  if (!accel.begin()) {
    Serial.println("No ADXL345 detected.");
    while (1);
  }

  accel.setRange(ADXL345_RANGE_2_G); //sets the sensitivity which can be selected
based on the application

  pinMode(dcmotorPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);

  pinMode(joyButtonPin, INPUT_PULLUP); //button pin mode using an inbuilt pullup
resistor

  speedServo.attach(servoPin);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  myStepper.setSpeed(60);  // RPM
}

// Sends 8 cycles of 40KHz sound frequency and uses the time this wave took to
```

```
bounce back and reach the sensor to calculate and return distance
long getDistance() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH);
  return duration * 0.034 / 2;
}

//Main loop which handles all the required skateboard functionalities
void loop() {
  // --- Mode switching via joystick button ---
  int buttonState = digitalRead(joyButtonPin);

  //Detects the button press and switches the drone mode, also prints out the
actual current mode
  if (buttonState == LOW && lastButtonState == HIGH) {
    droneMode = !droneMode;
    Serial.print("Drone mode: ");
    Serial.println(droneMode ? "ON" : "OFF");
    delay(200);
  }
  lastButtonState = buttonState; //updates the last button state which can then be
used to detect next button press

  //Read accelerometer data and store it in 'event' variable
  sensors_event_t event;
  accel.getEvent(&event);

  //Assigning the red accelerometer readings to variables
  float ax = event.acceleration.x;
  float ay = event.acceleration.y;
  float az = event.acceleration.z;
  float totalAccel = sqrt(ax*ax + ay*ay + az*az);

  // --- Fall/impact detection ---
  // As soon as the total acceleration crosses any predefined threshold values, it
is registered as a fall and hence
  if (totalAccel < lowerThreshold || totalAccel > upperThreshold) {
    analogWrite(dcmotorPin, 0); //DC motor will be stopped
    speedServo.write(0); //Servo motor displays 0 speed
    displayDigitOn7Segment(0); //7 segment display shows 0 digit
    Serial.print("Fall or impact detected!");
    while (true); //all functionalities are hereby stopped until and unless the
arduino is reset
  }

  // --------------------- SPEED CONTROL ---------------------
```

```
  // Adjust speed based on mode: joystick (droneMode) or tilt (default)
  int speed;
  //enters the loop only when drone mode is active
  if (droneMode) {
    int joyVal = analogRead(joyYPin); // y-axis controls speed

    // Increase or decrease speed based on y axis joystick movement
    if (joyVal > joyCenter + joyDeadzone) {
      currentSpeed += speedStep;
    } else if (joyVal < joyCenter - joyDeadzone) {
      currentSpeed -= speedStep;
    }
    //Changing the stepper angle based on x movement of joystick to simulate
steering
    int joyX = analogRead(joyXPin);
    if (abs(joyX - joyCenter) > joyDeadzone) {
      if (joyX > joyCenter) {
        myStepper.step(10);  // Turn right
      } else {
        myStepper.step(-10); // Turn left
      }
    }
    currentSpeed = constrain(currentSpeed, 0, 255); // limit speed between 0 and
255
    speed = currentSpeed;
  }


    //enters this loop only when drone mode is not active
   else {
    float x = ax / 9.81; //converting the obtained values into g's
    //increasing or decreasing the speed based on previously defined tilt threshold
    if (x > tiltThreshold) {
      currentSpeed += speedStep;
    } else if (x < -tiltThreshold) {
      currentSpeed -= speedStep;
    }
    currentSpeed = constrain(currentSpeed, 0, 255);
    speed = currentSpeed;
  }

  // --------------------- DISTANCE PROPORTIONAL OBSTACLE BRAKING
---------------------
  // Reduce speed if obstacle detected within 20cm
  long distance = getDistance();
  if (distance < 20) {
    int brakeFactor = map(distance, 0, 20, speed, 0); //map function used based on
distance to simulate proprotional braking
    speed = constrain(brakeFactor, 0, 255);
  }
```

```
  analogWrite(dcmotorPin, speed); //writing speed to DC motor using a PWM pin

  int displayDigit = map(speed, 0, 255, 0, 9); //mapping a speed value (from o to
9) based on actual speed (0 to 255)
  displayDigitOn7Segment(displayDigit);//displaying the mapped speed

  int angle = map(speed, 0, 255, 0, 180); //mapping a speed value angle (from 0 to
180) based on actual speed (0 to 255)
  speedServo.write(angle);// functioning as a speedometer


  // ---------------------- Print statements for debugging ----------------------
  Serial.println(
    String(" Speed: ") + speed +
    " Digit: " + displayDigit +
    " Servo: " + angle +
    " Accel: " + totalAccel +
    " Dist: " + distance
  );

  delay(100);
}

//Displays a single digit (0-9) on the 7-segment display via shift register.
void displayDigitOn7Segment(int digit) {
  digit = constrain(digit, 0, 9);
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, digitSegments[digit]);
  digitalWrite(latchPin, HIGH);
}
```