SafeLanding: An Intelligent Airbag System for Automated Fall Detection and Protection using Machine Learning and Internet-of-Things (IoT)

Richard Lin[1] and Yu Sun[2]

[1]Phillips Academy Andover, 180 Main St, Andover, MA 01810
[2]California State Polytechnic University,
Pomona, CA, 91768, Irvine, CA 92620

Abstract

In recent years, the world has been undergoing a drastic change in its age demographic due to an off balance caused by decreasing birth rates and an increase in the elderly population [1]. While 8.5% of the global population were elders in 2015, studies show that this number will hit 17% by 2050. This project will focus on the efficiency of automatic fall detection and contribute to the evolution of fall protection [2], both within elders and the general population. Through our conducted work, we have developed a wearable device capable of efficient fall detection and transmission.

Keywords

Machine Learning, Fall detection, Mobile APP.

1. Introduction

For such a growing demographic, one of the leading causes of injury and death is falling. The CDC estimates that 36 million falls happen each year among elders, and falling accounts for 95% of hip fractures. Fall mortality rates grew around 30% from 2008 to 2018 and steadily remain on the rise. Therefore, reliable and efficient fall detection is crucial to be developed [3]. While falling presents itself as a persistent issue, adequate steps have not been taken to prevent these threats and integrate efficient solutions into everyday life. Fall prevention devices and aids are not frequently used by elder populations, yet they make a substantial difference in the current amount of harm done and lives lost. There has been a growing incentive to keep elders active, so developing these devices is necessary.

Many existing techniques for fall detection utilize accelerometers to measure the acceleration of the user. However, these proposals falsely assume that acceleration is the only dimension of a person's falling motion [6]. Therefore, these systems may mistake other normal activities (such as running, jumping, or sitting) as falls. Most methods do not pull from an adequate number of sources of information, at times lacking the potentially useful implementation of gyroscopic sensors (measuring angular acceleration), magnetometers (measuring magnetic field strength and orientation), and GPS data to name a few.

Furthermore, some existing methods use inefficient means for detection. A study in 2014, although only a proof of concept, uses a smartphone sensor placed in a shirt pocket. This technique may be impractical for a user to replicate, and the data collected may be inaccurate due to the unstable placement of the phone.

Other studies occasionally rely on tracking the final body position of the user in order to register a fall in the system. A project back in 2009 first defines falls as the “unintentional transition to the lying posture,” where the intention is determined by accelerometer and gyroscopic data. While this can be used as a secondary step to verify a fall, systems that rely on this as a first step detection are slower and more complex to process. By requiring the collection of the user’s start and end positions in order to complete the first stage of fall detection, these processes are inherently performing after the fall has already taken place.

Finally, there are not enough efforts that are made towards the user experience of fall detection/prevention. Current fall prevention devices are primarily bulky, unaesthetic, expensive, and/or difficult to operate and reuse. We hope to start bridging this disconnect in this paper.

In this paper, we utilize and manipulate accelerometer and gyroscopic data to detect falls, as this combination of data provides a balance between simplicity and accuracy. Our system follows the similar lines of research, in that the fall is detected when the sensor surpasses a certain value threshold that is abnormal for other human behaviors.

Our process has a number of beneficial features. First, an efficient sensor is used, as the coin-sized device is placed discreetly on the front right hip. The placement of the sensor is used to streamline the data collection process and prioritizes the user’s comfortability. A number of existing research requires complex sensor systems that provide minimal to no gain in accuracy and/or functionality. Furthermore, we created an app (paired to the sensor) that is directed towards elders’ primary caretakers, which includes features such as a live detection display and a fall history log. Caretakers are notified when the user falls, and are able to dial emergency services only if needed. Therefore, we believe our method prioritizes efficiency and minimalism while remaining accurate.

In two scenarios, we demonstrate how the above combination of techniques benefits the efficiency of the device. First, we show the merit to our approach by conducting an extensive number of falls after developing our algorithm. Second, we interviewed a collection of seniors that provided feedback on what features would be useful to have on the app.

The rest of the paper is organized as follows: Section 2 gives the details on the challenges that we faced during our algorithm experimentation and app integration; Section 3 outlines our solutions to the corresponding challenges mentioned prior; Section 4 presents the relevant experiments and the field interviews we conducted, followed by an acknowledgement of related and noteworthy works in Section 5. Finally, Section 6 gives concluding remarks and discusses future areas of expansion for our project.

2. CHALLENGES

Here’s a brief overview of the common concerns we have identified that one may face in a fall situation.

2.1. User capabilities

Although falling is a potentially life-threatening action, user functionality is still crucial to any system looking to be developed into a product. When elders (or anyone) fall, they are often not able to recover on their own from their position on the ground. Therefore, most elders (who are in need of fall detection the most) have a caretaker or primary contact through which they are monitored for their safety [5]. In this system, caretakers are then the ones responsible for caring for their subject(s), whether in the case of a fall or other injuries. It is crucial for these people to be notified if a fall were to occur so that they may take the necessary further steps to ensure the safety of their subjects. In this system, since fall detection data needs to be processed and updated quickly, simplified models with fewer amounts of steps should be used.

Furthermore, tracking the history of falls of a user can be beneficial for medical practitioners, as it may reveal patterns to a user's behavior that could lead to properly diagnosing medical issues.

2.2. Device functionality and experience

For the device, fall detection devices should prioritize the wearer's experience. Some devices on the market are bulky on the body and heavy to wear [7]. Others are invasive to the user's daily activities, placed on parts of the body that may inhibit motion or cause discomfort. An optimal device is one that is lightweight and relatively small, as there would be little to no drawbacks in attaching such a device to the user.

On top of this, the device should efficiently be connected to the main controller app. The caretaker app should only display the necessary information needed to deduce the best course of action, such as the time of the most recent fall.

2.3. Affordability and accuracy of equipment

Although fall detection may be a tool that saves many lives, that does not mean it has to require expensive technology. While some state-of-the-art devices detect falls reliably, such as the apple watch, they boast a high price point due to its combination of other features. Affordable sensors are more than capable (in build and technology) of gathering accurate fall data. Some previous research has also required the usage of multiple sensors positioned on various points on the body. However, the results provide no substantial increase in accuracy as compared to single-sensor systems.

3. SOLUTION

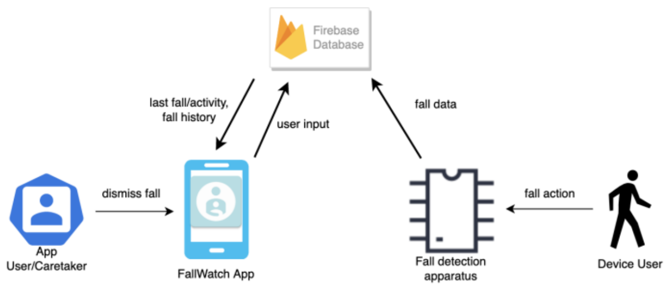


Figure 1. Overview of the solution

FallWatch is an environment that consists of two primary functionalities: the fall detection apparatus and the paired app. The apparatus is a junction of a QT PY ESP32-S3 controller chip linked to a 9-DoF (Degree of Freedom) board, allowing high-quality motion detection. On the board are the LSM6DSOX, the accelerometer and gyroscope sensor, and the LIS3MDL, a magnetometer. The accelerometer and gyroscope sensors have a poll rate of up to 6.7KHz, more than capable of updating the instantaneous data needed in fall scenarios. The accelerometer can measure up to +-16g of force, while the gyroscope +-2000dps (degrees per second).

The environment is built for efficiency. Once a fall is noted from the apparatus through the detection algorithm, the board updates a real-time database [8]. On the other hand, the FallWatch app is also connected to the database and is updated upon a fall. App users then have the option to dismiss and acknowledge the fall, however the sensors still remain active if the user chooses not to do so.

3.2.1 Hardware and Sensors

- Materials and component
- Connection diagrams
- Coding IDE and environment

3.2.2 Detection Algorithm

- Overview
- Diagram
- Code excerpt
- Explanation

3.2.3 Database and Communication

- Firebase introduction
- Pushing the data to Firebase
- Code excerpt

3.2.4 Mobile App

- For each Screen:
- Screenshot
- Code excerpt
- Explanation

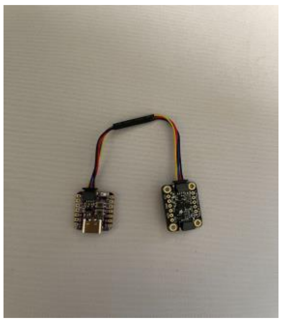


Figure 2. Image of device (sensor and controller boards)

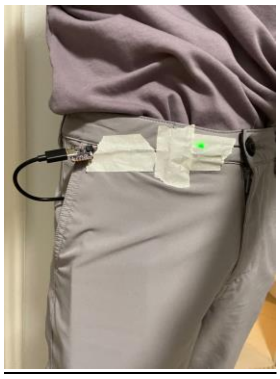
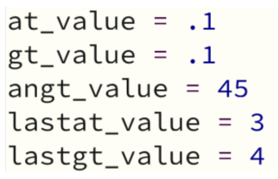


Figure 3. Image of device test position

The controller board (left) and the sensor board (right) are connected via an i2c cable. The dual device is then powered through the controller's USB-C port (with a battery pack or some other source). Both the device and the app are connected to the database over Wi-Fi connection. The sensor is then attached right side up to the hip on the right side of the front of the user's body, in the location where a belt would be worn.

Detection Algorithm [9]

Our system tracks the linear and angular acceleration to process the event of a fall. There are five values that go are used in the algorithm, each representing a threshold for a fall to be detected.

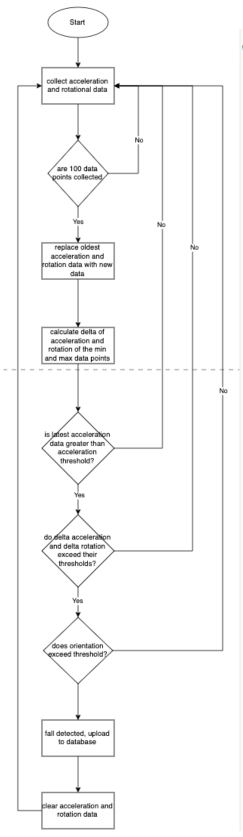


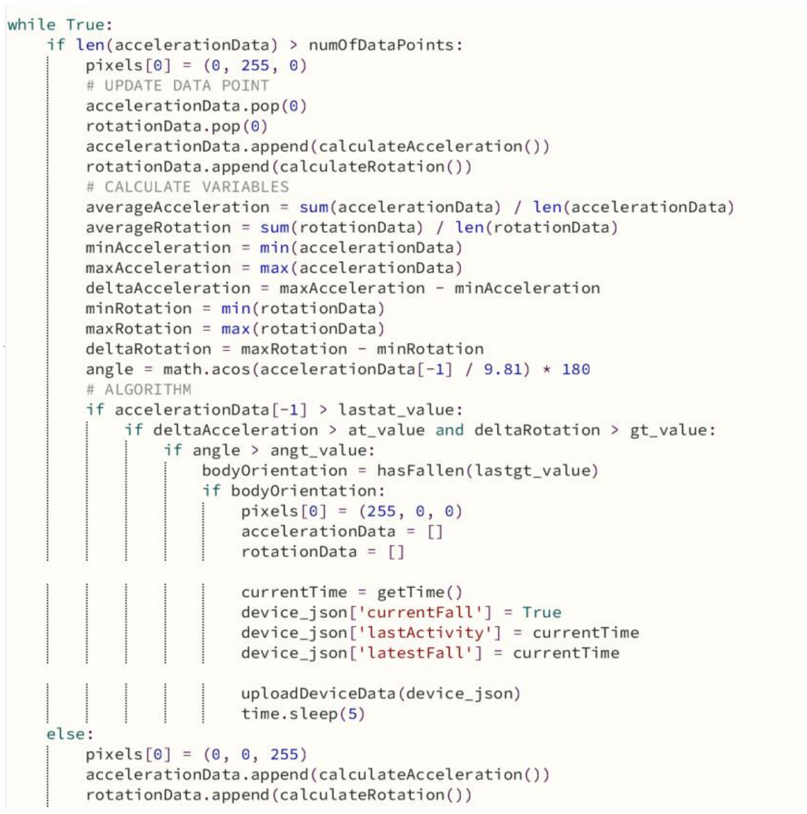
```
at_value = .1
gt_value = .1
angt_value = 45
lastat_value = 3
lastgt_value = 4
```

Figure 4. Values

"at_value" and "gt_value" are the thresholds for the delta between the highest and the lowest values of the last 100 linear and rotational acceleration data points. Essentially, a fall would register if there is a great enough change in both accelerations to surpass these values. "angt_value" is the angular threshold that the device must surpass, indicating the user's change in body orientation. "lastat_value" and "lastgt_value" are the linear and rotational acceleration thresholds for the last collected value of acceleration, to confirm whether the user is still in the process of accelerating due to a fall.

The algorithm is then produced with these values established. The following is a flowchart diagram of the algorithm (in the flowchart, "acceleration" is assumed to be linear):


Start
collect acceleration and rotational data
are 100 data points collected
No
Yes
replace oldest acceleration and rotation data with new data
calculate delta of acceleration and rotation of the min and max data points
No
No
No
is latest acceleration data greater than acceleration threshold?
Yes
do delta acceleration and delta rotation exceed their thresholds?
Yes
does orientation exceed threshold?
fall detected, upload to database
clear acceleration and rotation data



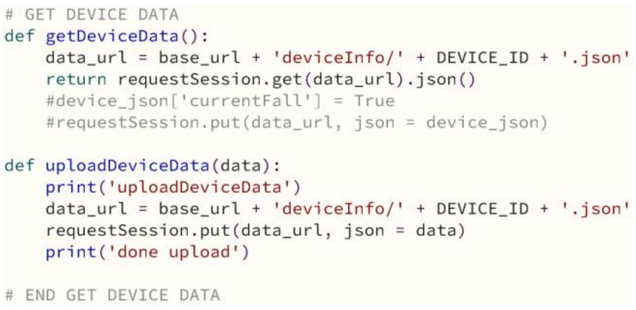
```
while True:
    if len(accelerationData) > numOfDataPoints:
        pixels[0] = (0, 255, 0)
        # UPDATE DATA POINT
        accelerationData.pop(0)
        rotationData.pop(0)
        accelerationData.append(calculateAcceleration())
        rotationData.append(calculateRotation())
        # CALCULATE VARIABLES
        averageAcceleration = sum(accelerationData) / len(accelerationData)
        averageRotation = sum(rotationData) / len(rotationData)
        minAcceleration = min(accelerationData)
        maxAcceleration = max(accelerationData)
        deltaAcceleration = maxAcceleration - minAcceleration
        minRotation = min(rotationData)
        maxRotation = max(rotationData)
        deltaRotation = maxRotation - minRotation
        angle = math.acos(accelerationData[-1] / 9.81) * 180
        # ALGORITHM
        if accelerationData[-1] > lastat_value:
            if deltaAcceleration > at_value and deltaRotation > gt_value:
                if angle > angt_value:
                    bodyOrientation = hasFallen(lastgt_value)
                    if bodyOrientation:
                        pixels[0] = (255, 0, 0)
                        accelerationData = []
                        rotationData = []

                        currentTime = getTime()
                        device_json['currentFall'] = True
                        device_json['lastActivity'] = currentTime
                        device_json['latestFall'] = currentTime

                        uploadDeviceData(device_json)
                        time.sleep(5)
    else:
        pixels[0] = (0, 0, 255)
        accelerationData.append(calculateAcceleration())
        rotationData.append(calculateRotation())
```

Figure 5. Flowchart diagram of the algorithm

First, the device collects a set of linear and rotational acceleration data points. Once the system has detected enough, it replaces the oldest acceleration data with a new data point. These data points are calculated from the “calculateAcceleration” and “calculateRotation” functions, which take the square root of the sum of the square of each dimension (ax2+ ay2+az2). From there, the algorithm takes the largest and smallest values in the new set and calculates the difference between them. Then, the system checks to see if the thresholds have been surpassed. It first determines whether the most recent linear acceleration exceeds the defined threshold. If so, it checks if the deltas of the linear and rotational accelerations are greater than their thresholds. If this is confirmed, the angle is checked by first determining whether the user has fallen greater than the angle threshold set, and then confirms that the most recent angular acceleration data point exceeds its threshold. At last, a fall is detected and uploaded to the database, and all data points are cleared while the device remains active.

Database and Communication

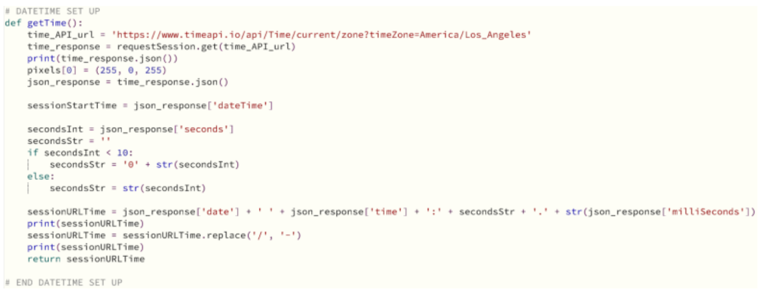
For exemplary purposes, Firebase is used to store user and device information, updated fall data, and fall histories. There are two main segments concerning the database: the manipulation of device data and timestamps.



```
# GET DEVICE DATA
def getDeviceData():
    data_url = base_url + 'deviceInfo/' + DEVICE_ID + '.json'
    return requestSession.get(data_url).json()
    #device_json['currentFall'] = True
    #requestSession.put(data_url, json = device_json)

def uploadDeviceData(data):
    print('uploadDeviceData')
    data_url = base_url + 'deviceInfo/' + DEVICE_ID + '.json'
    requestSession.put(data_url, json = data)
    print('done upload')

# END GET DEVICE DATA
```

Figure 6. Sending and receiving device data from database

In both receiving and uploading device data to the database, a URL is created [10]. Since a unique six-character (three letter three number) ID is assigned to each device and stored in the database, this ID is used to generate a unique link for the data of that specific device. With this URL labeled “data_url,” data can be received or updated with the respective functions shown above.

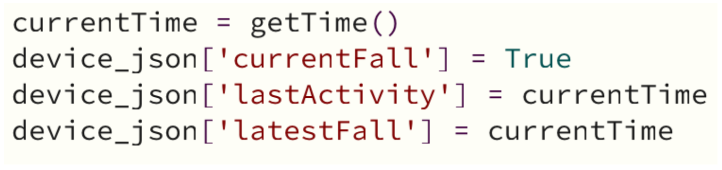


```
# DATETIME SET UP
def getTime():
    time_API_url = 'https://www.timeapi.io/api/Time/current/zone?timeZone=America/Los_Angeles'
    time_response = requestSession.get(time_API_url)
    print(time_response.json())
    pixels[0] = (255, 0, 255)
    json_response = time_response.json()

    sessionStartTime = json_response['dateTime']

    secondsInt = json_response['seconds']
    secondsStr = ''
    if secondsInt < 10:
        secondsStr = '0' + str(secondsInt)
    else:
        secondsStr = str(secondsInt)

    sessionURLTime = json_response['date'] + ' ' + json_response['time'] + ':' + secondsStr + '.' + str(json_response['milliSeconds'])
    print(sessionURLTime)
    sessionURLTime = sessionURLTime.replace('/', '-')
    print(sessionURLTime)
    return sessionURLTime

# END DATETIME SET UP
```

Figure 7. Structuring date and time to be uploaded to database

To track the time of falls, time is implemented via the timeapi.io live website. The time received must be reformatted before being uploaded to the database.

Device data and time are both used in the outcomes of the algorithm. For instance, when the system detects a fall, the “device_json” updates its values according to the fall event that occurred. The fall is set to true, and both the time of the last activity and latest fall are set to the present time. These values are ultimately read by the app, which updates according to the database.



```
currentTime = getTime()
device_json['currentFall'] = True
device_json['lastActivity'] = currentTime
device_json['latestFall'] = currentTime
```

Figure 8. Updating database data in fall occurrence

Mobile App

The mobile app is used as a live indication for when a fall has occurred [15]. The following are the key features of the app:

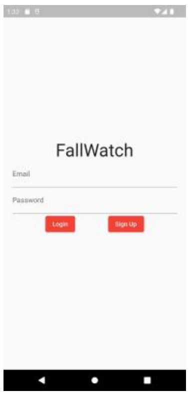
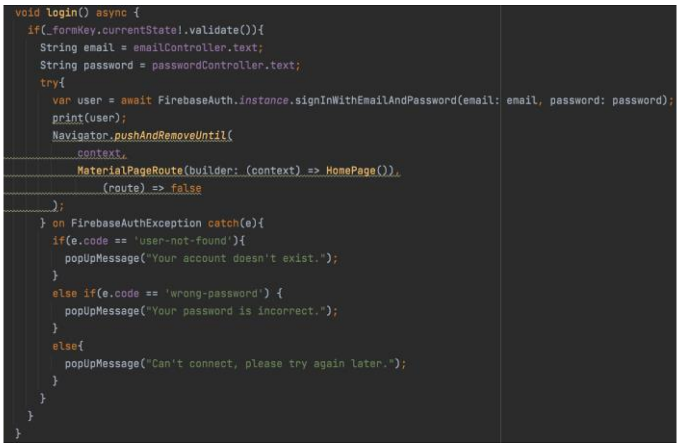


Figure 9. Login screen



```
void login() async {
  if(_formKey.currentState!.validate()){
    String email = emailController.text;
    String password = passwordController.text;
    try{
      var user = await FirebaseAuth.instance.signInWithEmailAndPassword(email: email, password: password);
      print(user);
      Navigator.pushAndRemoveUntil(
          context,
          MaterialPageRoute(builder: (context) => HomePage()),
              (route) => false
      );
    } on FirebaseAuthException catch(e){
      if(e.code == 'user-not-found'){
        popUpMessage("Your account doesn't exist.");
      }
      else if(e.code == 'wrong-password') {
        popUpMessage("Your password is incorrect.");
      }
      else{
        popUpMessage("Can't connect, please try again later.");
      }
    }
  }
}
```

Figure 10. Login screen code excerpt

The login interface is used to register the user in the database. Then, the fall device can be connected to the account via the unique six character device ID.

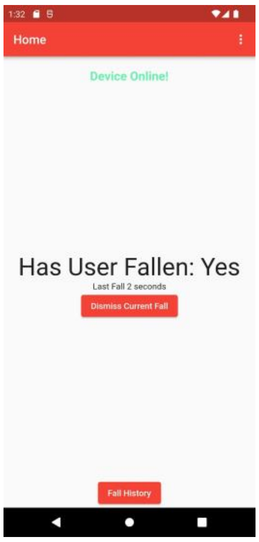
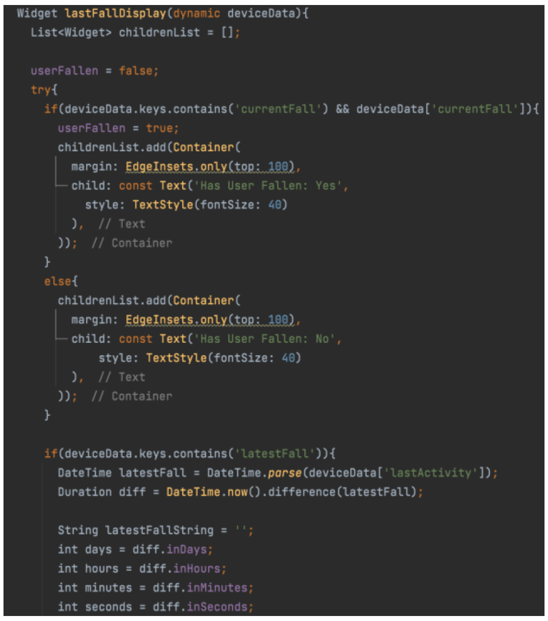


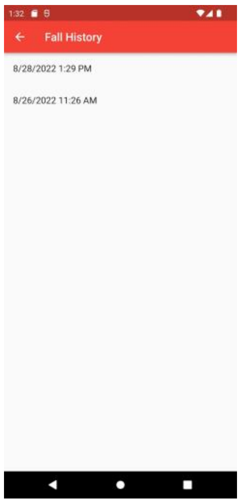
Figure 11. Homepage screen



```
Widget lastFallDisplay(dynamic deviceData){
  List<Widget> childrenList = [];

  userFallen = false;
  try{
    if(deviceData.keys.contains('currentFall') && deviceData['currentFall']){
      userFallen = true;
      childrenList.add(Container(
        margin: EdgeInsets.only(top: 100),
        child: const Text('Has User Fallen: Yes',
          style: TextStyle(fontSize: 40)
        ),  // Text
      ));  // Container
    }
    else{
      childrenList.add(Container(
        margin: EdgeInsets.only(top: 100),
        child: const Text('Has User Fallen: No',
            style: TextStyle(fontSize: 40)
        ),  // Text
      ));  // Container
    }

    if(deviceData.keys.contains('latestFall')){
      DateTime latestFall = DateTime.parse(deviceData['lastActivity']);
      Duration diff = DateTime.now().difference(latestFall);

      String latestFallString = '';
      int days = diff.inDays;
      int hours = diff.inHours;
      int minutes = diff.inMinutes;
      int seconds = diff.inSeconds;
```

Figure 12. Homepage code excerpt (detecting and displaying information about latest fall)

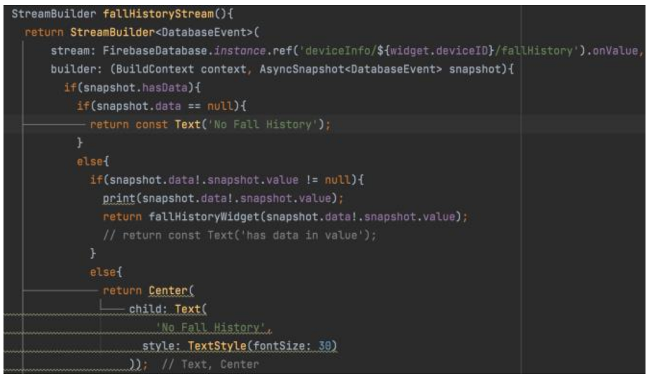
The homepage includes a few key components. At the top, the status of the device is displayed as whether the device is online and connected or offline. The center widget updates if the device detects a fall and shows the time elapsed since the last fall. The "Has User Fallen" status changes back to "No" after two minutes of no fall activity. There is also a button for the app user to dismiss the current fall, which manually reverts the status. At the bottom, there is a button for the fall history, which is as follows:



8/28/2022 1:29 PM

8/26/2022 11:26 AM

Figure 13. Fall history screen



```
StreamBuilder fallHistoryStream(){
  return StreamBuilder<DatabaseEvent>(
      stream: FirebaseDatabase.instance.ref('deviceInfo/${widget.deviceID}/fallHistory').onValue,
      builder: (BuildContext context, AsyncSnapshot<DatabaseEvent> snapshot){
        if(snapshot.hasData){
          if(snapshot.data == null){
            return const Text('No Fall History');
          }
          else{
            if(snapshot.data!.snapshot.value != null){
              print(snapshot.data!.snapshot.value);
              return fallHistoryWidget(snapshot.data!.snapshot.value);
              // return const Text('has data in value');
            }
            else{
              return Center(
                child: Text(
                    'No Fall History',
                  style: TextStyle(fontSize: 30)
                )); // Text, Center
```

Figure 14. Fall history screen code excerpt (retrieving data from database)

Here, a simple fall history log is displayed. The app gathers the device's past falls and displays them in a list ordered from most recent downwards.

4. Experiment

- trials based on algo values, divided into 9 tests each (3 for each type of fall)
- data table and bar chart for success rate

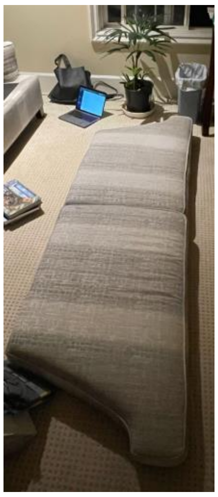
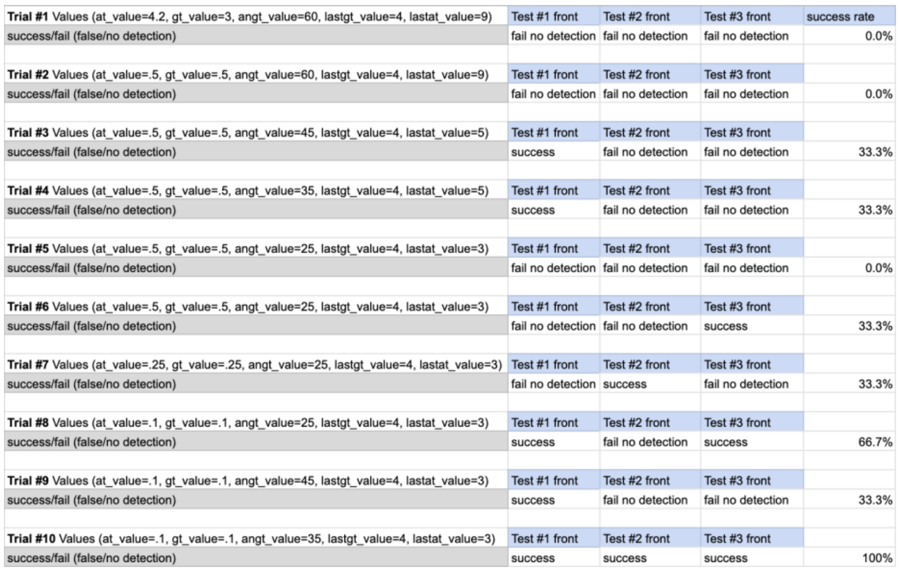


Figure 15. Fall area

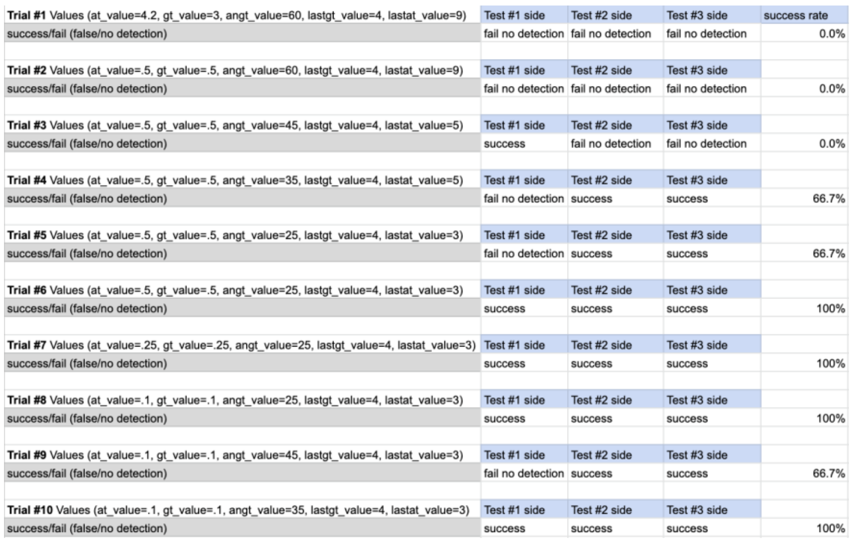
4.1. Experiment 1

To refine the algorithm, we tested falls until the set values produced accurate results. The device was plugged into a constant battery and taped on through the belt loop. An 8 foot fall landing area was set up and cushioned. Three mini-experiments were conducted in the setup, one for each type of fall (forward, sideways, backward). In each experiment, three falls were done for each set of threshold values, and the success of the device was recorded. Thresholds were then altered according to the sensitivity of the sensor for each subsequent set of tests. In total, 10 sets of values were tested.



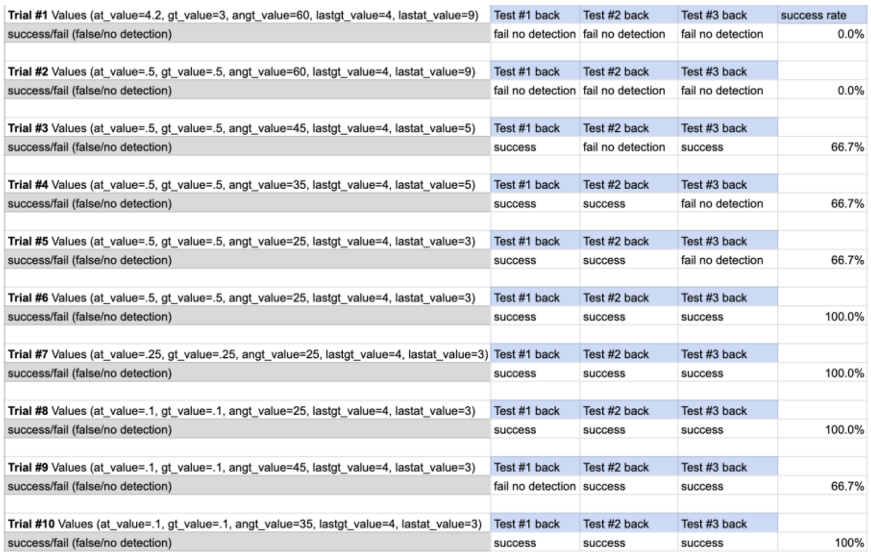
Trial #1 Values (at_value=4.2, gt_value=3, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 front	Test #2 front	Test #3 front	success rate
success/fail (false/no detection)	fail no detection	fail no detection	fail no detection	0.0%
Trial #2 Values (at_value=.5, gt_value=.5, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 front	Test #2 front	Test #3 front	
success/fail (false/no detection)	fail no detection	fail no detection	fail no detection	0.0%
Trial #3 Values (at_value=.5, gt_value=.5, angt_value=45, lastgt_value=4, lastat_value=5)	Test #1 front	Test #2 front	Test #3 front	
success/fail (false/no detection)	success	fail no detection	fail no detection	33.3%
Trial #4 Values (at_value=.5, gt_value=.5, angt_value=35, lastgt_value=4, lastat_value=5)	Test #1 front	Test #2 front	Test #3 front	
success/fail (false/no detection)	success	fail no detection	fail no detection	33.3%
Trial #5 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 front	Test #2 front	Test #3 front	
success/fail (false/no detection)	fail no detection	fail no detection	fail no detection	0.0%
Trial #6 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 front	Test #2 front	Test #3 front	
success/fail (false/no detection)	fail no detection	fail no detection	success	33.3%
Trial #7 Values (at_value=.25, gt_value=.25, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 front	Test #2 front	Test #3 front	
success/fail (false/no detection)	fail no detection	success	fail no detection	33.3%
Trial #8 Values (at_value=.1, gt_value=.1, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 front	Test #2 front	Test #3 front	
success/fail (false/no detection)	success	fail no detection	success	66.7%
Trial #9 Values (at_value=.1, gt_value=.1, angt_value=45, lastgt_value=4, lastat_value=3)	Test #1 front	Test #2 front	Test #3 front	
success/fail (false/no detection)	success	fail no detection	fail no detection	33.3%
Trial #10 Values (at_value=.1, gt_value=.1, angt_value=35, lastgt_value=4, lastat_value=3)	Test #1 front	Test #2 front	Test #3 front	
success/fail (false/no detection)	success	success	success	100%

Figure 16. Data for experiment #1 (forward fall)



Trial #1 Values (at_value=4.2, gt_value=3, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 side	Test #2 side	Test #3 side	success rate
success/fail (false/no detection)	fail no detection	fail no detection	fail no detection	0.0%
Trial #2 Values (at_value=.5, gt_value=.5, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 side	Test #2 side	Test #3 side	
success/fail (false/no detection)	fail no detection	fail no detection	fail no detection	0.0%
Trial #3 Values (at_value=.5, gt_value=.5, angt_value=45, lastgt_value=4, lastat_value=5)	Test #1 side	Test #2 side	Test #3 side	
success/fail (false/no detection)	success	fail no detection	fail no detection	0.0%
Trial #4 Values (at_value=.5, gt_value=.5, angt_value=35, lastgt_value=4, lastat_value=5)	Test #1 side	Test #2 side	Test #3 side	
success/fail (false/no detection)	fail no detection	success	success	66.7%
Trial #5 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 side	Test #2 side	Test #3 side	
success/fail (false/no detection)	fail no detection	success	success	66.7%
Trial #6 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 side	Test #2 side	Test #3 side	
success/fail (false/no detection)	success	success	success	100%
Trial #7 Values (at_value=.25, gt_value=.25, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 side	Test #2 side	Test #3 side	
success/fail (false/no detection)	success	success	success	100%
Trial #8 Values (at_value=.1, gt_value=.1, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 side	Test #2 side	Test #3 side	
success/fail (false/no detection)	success	success	success	100%
Trial #9 Values (at_value=.1, gt_value=.1, angt_value=45, lastgt_value=4, lastat_value=3)	Test #1 side	Test #2 side	Test #3 side	
success/fail (false/no detection)	fail no detection	success	success	66.7%
Trial #10 Values (at_value=.1, gt_value=.1, angt_value=35, lastgt_value=4, lastat_value=3)	Test #1 side	Test #2 side	Test #3 side	
success/fail (false/no detection)	success	success	success	100%

Figure 17. Data for experiment #2 (sideways fall)

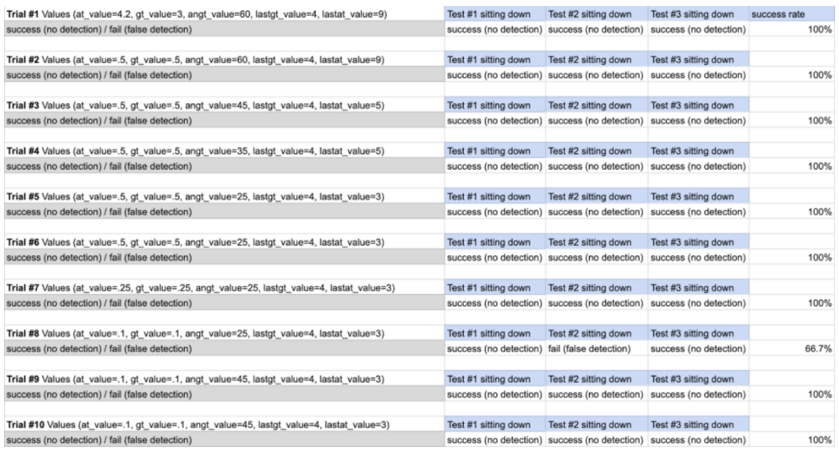


Trial #1 Values (at_value=4.2, gt_value=3, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 back	Test #2 back	Test #3 back	success rate
success/fail (false/no detection)	fail no detection	fail no detection	fail no detection	0.0%
Trial #2 Values (at_value=.5, gt_value=.5, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 back	Test #2 back	Test #3 back	
success/fail (false/no detection)	fail no detection	fail no detection	fail no detection	0.0%
Trial #3 Values (at_value=.5, gt_value=.5, angt_value=45, lastgt_value=4, lastat_value=5)	Test #1 back	Test #2 back	Test #3 back	
success/fail (false/no detection)	success	fail no detection	success	66.7%
Trial #4 Values (at_value=.5, gt_value=.5, angt_value=35, lastgt_value=4, lastat_value=5)	Test #1 back	Test #2 back	Test #3 back	
success/fail (false/no detection)	success	success	fail no detection	66.7%
Trial #5 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 back	Test #2 back	Test #3 back	
success/fail (false/no detection)	success	success	fail no detection	66.7%
Trial #6 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 back	Test #2 back	Test #3 back	
success/fail (false/no detection)	success	success	success	100.0%
Trial #7 Values (at_value=.25, gt_value=.25, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 back	Test #2 back	Test #3 back	
success/fail (false/no detection)	success	success	success	100.0%
Trial #8 Values (at_value=.1, gt_value=.1, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 back	Test #2 back	Test #3 back	
success/fail (false/no detection)	success	success	success	100.0%
Trial #9 Values (at_value=.1, gt_value=.1, angt_value=45, lastgt_value=4, lastat_value=3)	Test #1 back	Test #2 back	Test #3 back	
success/fail (false/no detection)	fail no detection	success	success	66.7%
Trial #10 Values (at_value=.1, gt_value=.1, angt_value=35, lastgt_value=4, lastat_value=3)	Test #1 back	Test #2 back	Test #3 back	
success/fail (false/no detection)	success	success	success	100%

Figure 18. Data for experiment #3 (backward fall)

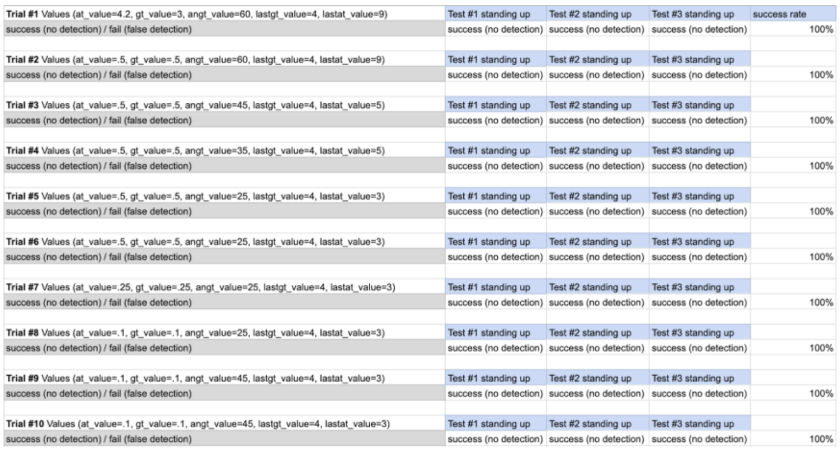
4.2. Experiment 2

In a second set of experiments, we needed to test for the event of false positives that mistake normal actions as falls. To do so, we tested each set of threshold values for three types of normal actions: standing up, sitting down, and walking. Each mini-experiment is structured the same way as the ones prior. The following are the trials and success rates for each action.



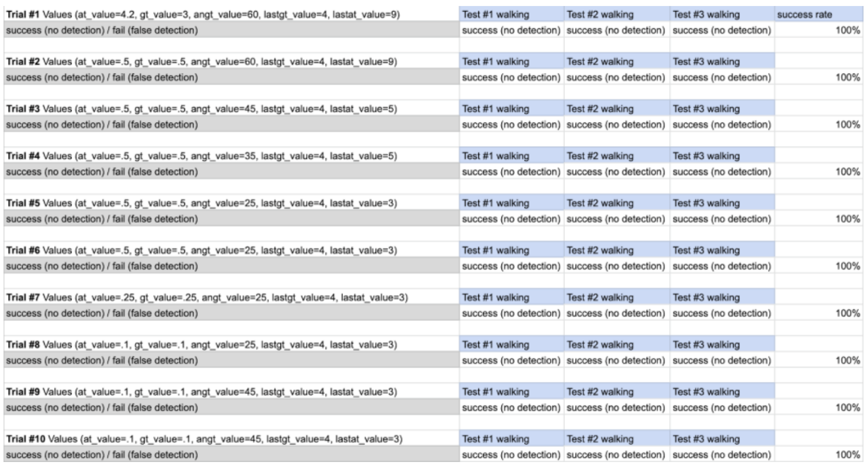
Trial #1 Values (at_value=4.2, gt_value=3, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	success rate
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #2 Values (at_value=.5, gt_value=.5, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #3 Values (at_value=.5, gt_value=.5, angt_value=45, lastgt_value=4, lastat_value=5)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #4 Values (at_value=.5, gt_value=.5, angt_value=35, lastgt_value=4, lastat_value=5)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #5 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #6 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #7 Values (at_value=.25, gt_value=.25, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #8 Values (at_value=.1, gt_value=.1, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	
success (no detection) / fail (false detection)	success (no detection)	fail (false detection)	success (no detection)	66.7%
Trial #9 Values (at_value=.1, gt_value=.1, angt_value=45, lastgt_value=4, lastat_value=3)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #10 Values (at_value=.1, gt_value=.1, angt_value=45, lastgt_value=4, lastat_value=3)	Test #1 sitting down	Test #2 sitting down	Test #3 sitting down	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%

Figure 19. Data for experiment #4 (sitting down)



Trial #1 Values (at_value=4.2, gt_value=3, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 standing up	Test #2 standing up	Test #3 standing up	success rate
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #2 Values (at_value=.5, gt_value=.5, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 standing up	Test #2 standing up	Test #3 standing up	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #3 Values (at_value=.5, gt_value=.5, angt_value=45, lastgt_value=4, lastat_value=5)	Test #1 standing up	Test #2 standing up	Test #3 standing up	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #4 Values (at_value=.5, gt_value=.5, angt_value=35, lastgt_value=4, lastat_value=5)	Test #1 standing up	Test #2 standing up	Test #3 standing up	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #5 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 standing up	Test #2 standing up	Test #3 standing up	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #6 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 standing up	Test #2 standing up	Test #3 standing up	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #7 Values (at_value=.25, gt_value=.25, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 standing up	Test #2 standing up	Test #3 standing up	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #8 Values (at_value=.1, gt_value=.1, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 standing up	Test #2 standing up	Test #3 standing up	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #9 Values (at_value=.1, gt_value=.1, angt_value=45, lastgt_value=4, lastat_value=3)	Test #1 standing up	Test #2 standing up	Test #3 standing up	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #10 Values (at_value=.1, gt_value=.1, angt_value=45, lastgt_value=4, lastat_value=3)	Test #1 standing up	Test #2 standing up	Test #3 standing up	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%

Figure 20. Data for experiment #5 (standing up)



Trial #1 Values (at_value=4.2, gt_value=3, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 walking	Test #2 walking	Test #3 walking	success rate
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #2 Values (at_value=.5, gt_value=.5, angt_value=60, lastgt_value=4, lastat_value=9)	Test #1 walking	Test #2 walking	Test #3 walking	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #3 Values (at_value=.5, gt_value=.5, angt_value=45, lastgt_value=4, lastat_value=5)	Test #1 walking	Test #2 walking	Test #3 walking	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #4 Values (at_value=.5, gt_value=.5, angt_value=35, lastgt_value=4, lastat_value=5)	Test #1 walking	Test #2 walking	Test #3 walking	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #5 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 walking	Test #2 walking	Test #3 walking	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #6 Values (at_value=.5, gt_value=.5, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 walking	Test #2 walking	Test #3 walking	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #7 Values (at_value=.25, gt_value=.25, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 walking	Test #2 walking	Test #3 walking	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #8 Values (at_value=.1, gt_value=.1, angt_value=25, lastgt_value=4, lastat_value=3)	Test #1 walking	Test #2 walking	Test #3 walking	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #9 Values (at_value=.1, gt_value=.1, angt_value=45, lastgt_value=4, lastat_value=3)	Test #1 walking	Test #2 walking	Test #3 walking	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%
Trial #10 Values (at_value=.1, gt_value=.1, angt_value=45, lastgt_value=4, lastat_value=3)	Test #1 walking	Test #2 walking	Test #3 walking	
success (no detection) / fail (false detection)	success (no detection)	success (no detection)	success (no detection)	100%

Figure 21. Data for experiment #6 (walking)

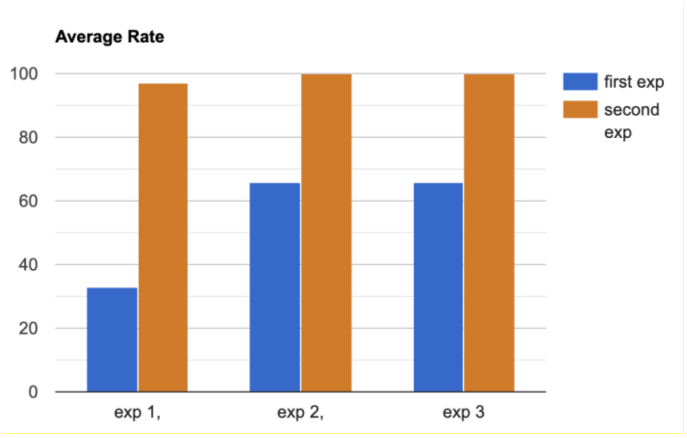


Figure 22. Average rate for 3 exp.

For our tests, the values of trial #10 (at_value = 0.1, gt_value = 0.1, angt_value = 35, lastgt_value = 4, lastat_value = 3) found the most success in both the fall and false test experiments. They were 100% accurate in the limited number of tests that we ran, and we acknowledge the low sample size as a potential pitfall. However, our results point out the fall detection capabilities of the tested sensors, which are meant to be affordable, compact, and noninvasive.

5. Related Work

This work, published in 2009 by Zhou and his team, contains a similar approach [11]. Zhou's system features two TEMPO (Technology-Enabled Medical Precision Observation) 3.0 nodes placed on the front chest and thigh. Both have tri-axial gyroscope and accelerometer sensors with adequate sensitivities. The work defines and tests for four types of motion: activities of daily life (ADL), fall-like motions, flat falls, and falls onto inclined surfaces. Both the featured work and our algorithm use similar methods for calculating combined acceleration and body orientation. Both systems set the same types of thresholds for linear acceleration, rotational acceleration, and angle. However, the usage of two sensors can both be beneficial and detrimental. While it may produce more data for the algorithm to process, we must keep in mind that the positions of these sensors are in areas of the body (the chest and thigh) that are subject to high amounts of motion. Meanwhile, the hip stays relatively aligned with the body orientation, which may be more reliable to base a system off of.

Research done by this team utilizes a smartphone fitted in the chest shirt pocket as the tri-axial accelerometer and gyroscope [12]. Average linear and rotational acceleration are calculated in similar ways (square root of sum of dimensions squared), as well as angle. Both system algorithms also use similar methods of analysis. However, comparisons between the last instantaneous data point and the entire recorded data set are not made, which may cause more false positives with non-fall actions. Once again, the placement of the sensor may not be as reliable, but the data seems to show a high success rate nonetheless.

This research discusses energy consumption and battery life in different sensors that are used for fall detection purposes [13]. It proposes two types of sensor nodes: those that are custom made for fall detection and others that are general purpose devices. This is a unique angle when analyzing the integrating of fall devices. Since falls are spontaneous and can occur at any given moment, batteries need to last a sufficient length of time.

6. Conclusions

This system uses a tri-axis accelerometer and gyroscope attached to the hip to detect falls [14]. The sensors and controller communicate directly with a database, which stores previous activity and device information. A simple app was then created to simulate a common caretaker-elder relationship, where the app is updated and notified upon the device's trigger. This app-device environment serves as a proof of concept for an example of a fall response system.

To detect a fall, thresholds for linear acceleration, rotational acceleration, and angular change are set so that upon surpassing all three in the order of the algorithm, a fall is registered.

Because our work prioritized affordability as a key value, accuracy may suffer from the use of less expensive sensors (however the data collected does not indicate such an issue). Also, we have only created a post-fall detection algorithm like much research currently published [4]. A different method would likely need to be used to increase the speed at which the fall is initially detected. Finally, a different data sample would be beneficial to collect, one that incorporates the elder population as test subjects.

With a plan to implement automatic response capabilities to the device, developing a machine learning algorithm may be the optimal approach to creating not only a fall detection system, but a fall prevention product. Although implementing an airbag deployment system seems intuitive (and has been experimented with), this design has not been readily available on the market due to regulation constraints. In the expansive market of fall injuries, innovative steps must be made to ensure the safety of countless lives in the future.

REFERENCES

[1] Hobbs, Frank B. "The elderly population." US Census Bureau Population Profile of the United States, http://www. census. gov/population/www/pop-profile/elderpop. html (2001).

[2] Ellis, J. Nigel. "What is Fall Protection?." ASSE Professional Development Conference and Exposition. OnePetro, 2000.

[3] Mubashir, Muhammad, Ling Shao, and Luke Seed. "A survey on fall detection: Principles and approaches." Neurocomputing 100 (2013): 144-152.

[4] Igual, Raul, Carlos Medrano, and Inmaculada Plaza. "Challenges, issues and trends in fall detection systems." Biomedical engineering online 12.1 (2013): 1-24.

[5] Wang, Xueyi, Joshua Ellul, and George Azzopardi. "Elderly fall detection systems: A literature survey." Frontiers in Robotics and AI 7 (2020): 71.

[6] Southern, W. Thomas, and Eric D. Jones. "Types of acceleration: Dimensions and issues." A nation deceived: How schools hold back America's brightest students 2 (2004): 5-12.

[7] Negahban, Arash, and Chih-Hung Chung. "Discovering determinants of users perception of mobile device functionality fit." Computers in Human Behavior 35 (2014): 75-84.

[8] Wedekind, Hartmut, and George Zoerntlein. "Prefetching in realtime database applications." ACM SIGMOD Record 15.2 (1986): 215-226.

[9] Pan, Jiapu, and Willis J. Tompkins. "A real-time QRS detection algorithm." IEEE transactions on biomedical engineering 3 (1985): 230-236.

[10] Berners-Lee, Tim, Larry Masinter, and Mark McCahill. Uniform resource locators (URL). No. rfc1738. 1994.

[11] Li, Qiang, et al. "Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information." 2009 Sixth International Workshop on Wearable and Implantable Body Sensor Networks. IEEE, 2009.

[12] Rakhman, Arkham Zahri, and Lukito Edi Nugroho. "Fall detection system using accelerometer and gyroscope based on smartphone." 2014 The 1st International Conference on Information Technology, Computer, and Electrical Engineering. IEEE, 2014.

[13] Gia, Tuan Nguyen, et al. "IoT-based fall detection system with energy efficient sensor nodes." 2016 IEEE Nordic Circuits and Systems Conference (NORCAS). IEEE, 2016.

[14] ㅔ에. "Physical activity recognition using a single tri-axis accelerometer." Proceedings of the world congress on engineering and computer science. Vol. 1. 2009.

[15] Joorabchi, Mona Erfani, Ali Mesbah, and Philippe Kruchten. "Real challenges in mobile app development." 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, 2013.