International Journal of Computer Applications (0975 - 8887)
Volume 1 - No.1, May 2024

Inertial navigation systems for phones without GPS

Ibrahim Alaiddin Abu Rached
Aalborg University
irache20@student.aau.dk

Karl Barnholdt Kragsaa
Aalborg University
kkrags20@student.aau.dk

Marcus Hasselgaard Skifter Jensen
Aalborg University
mhsj20@student.aau.dk

Rune Gøttsche Aagaard
Aalborg University
raagaa20@student.aau.dk

Sebastian Malthe Røn
Aalborg University
sm20@student.aau.dk

Tobias Martin Pihl
Aalborg University
tpihl20@student.aau.dk

ABSTRACT

In the modern world GPS is widely used as a tool for navigation by citizens. People can easily navigate from point A to point B and also gain information about shops and landmarks close to their current location. This is largely thanks to apps like Google Maps, which makes use of GPS to track user location [3]. One downside of GPS is that most map apps require a good connection to function properly. This paper examines how to use a phone to navigate in places where there is no GPS signal. The paper features comparisons of different approaches for inertial navigation and a proposal for how to improve the accuracy of navigation. The paper concludes that existing inertial navigation systems are insufficient in their accuracy to be used in real world scenarios. This is due to unhandled gyroscope drifting and bad accelerometer filtering for calculating distances. The solution proposed in this paper outperforms the accuracy of existing solutions and is able to draw longer and more complex routes which makes the proposed solution more viable in real world scenarios.

General Terms

MEMS (Micro Electro Mechanical Systems), IMU (Inertial Measurement Unit), INS (Inertial Navigation System), GPS (Global Positioning System), UI (User Interface).

Keywords

GPS-free navigation, inertial navigation, phone sensors, accelerometer, gyroscope, magnetometer, cave exploring.

1. INTRODUCTION

GPS is commonly used in combination with apps like Google Maps to allow easy navigation for users. This combination of user location and stored knowledge about the surroundings creates an effective tool for navigation. However, challenges arise in areas with no internet connection. While GPS can still function without an internet connection, the same is not necessarily true for a navigation app [3].

This paper examines how to draw maps in areas where there is no GPS signal. The paper discusses how this can be implemented in the context of drawing a map inside a cave. Traditionally, a cave explorer could tie a knot somewhere fixed outside the cave before entering, bring the other end of the rope with them inside and then follow the rope back to find their way out. However, this requires the explorer to bring a very long rope if the cave is large. This motivates us to draw maps in areas where there is no GPS signal and more specifically, study how to merely use a phone's sensors to find the way.

The method for calculating distances outlined by Germán Rodríguez et al. [5] explores how step detection can be used to calculate distances. This method uses a fixed step length and the number of detected steps to determine how far a person has moved. This approach has the disadvantage that their step detection procedure produces false positives (detected steps without walking). When walking longer routes, this error will grow larger and larger. Corina Kim Schindhelm, et al. [6] also explores how to calculate distances using the accelerometer. Their approach to calculating distances relies on filtering accelerometer data and using that filtered data to calculate speed by integrating the recorded accelerations. They also use gyroscope data to compute orientation with the purpose of drawing a map. However, the results are very vague in the sense that their only experiment is an attempt to draw a circle with a one meter radius.

To address the above limitations, we develop an app that utilizes a phone's gyroscope, accelerometer, and magnetometer for navigation in areas without GPS signals. This app is capable of drawing accurate maps in real time. Specifically, the study by Corina Kim Schindhelm et al. [6] indicates that the gyroscope is imprecise due to drifting. Therefore, our paper addresses how to implement a viable step detection mechanism that continuously computes velocity to calculate a variable step length, rather than a fixed one. In addition to this, our proposed solution compensate for gyroscope drifting, which allows for drawing longer and more precise routes. Experiments in real-world scenarios provide evidence that the proposed app can accurately map both long and short routes.

To summarize the contributions of this paper:

- To design a solution for navigating in areas where there is no GPS signal. This is achieved solely by using an accelerometer, a gyroscope and a magnetometer.
- To draw longer and more complex routes than circles, we implement compensation for gyroscope drifting and step detection with variable step length.
- The results presented in this paper show that our proposed solution can draw maps in real time on distances between 14 meters and 800 meters with a deviation in accuracy below 2%.

The rest of the paper is organized the following way: Section 2 provides an overview of related works within the field of inertial navigation. Section 3 provides information about accelerometers, gyroscopes, magnetometers and step detection tecniques. Section 4 goes into detail with the implementation of gravity compensation, distance calculation, compensation of gyroscope drifting and step detection. Section 5 includes the experiments, results and a discussion of the results. Lastly, section 6 lays out the conclusion and an overview of possible future works.

2. RELATED WORK

[2] researches using phone sensors and image-based positioning to improve the latency to accurately estimate position in cloud architectures. [8] examines various error sources that occur in MEMS accelerometers and gyroscopes. Furthermore, the paper applies techniques such as the Allan Variance, therein the Allan Deviation. These techniques are used to detect the characteristics of the noise in the signal from the various sensors. Lastly, the rectangular rule is used in the context for providing update equation for tracking the position of the device, in which the sensors are placed. [5] implements an inertial navigation system by using step counting through a simple peak valley algorithm over the sensor data and integrating it with a walking recognition algorithm to increase its robustness.

Corina Kim Schindhelm et al. [6] shares the same research focus as us. They specifically focus on Apple iPhones whereas we focused more generally on phones. As the research paper is from 2011, the phones used were an iPhone 3GS and an iPhone 4. The first phone only came equipped with an accelerometer while the iPhone 4 included a gyroscope as well. The paper noted that a lot of navigation systems rely on information from third-party providers for example GPS. This was an issue for privacy. IMU provides an alternative to this where all information about location is local and built-in sensors are used to get an idea about how a device is moving. This can then guarantee total privacy without relying on external parties. The paper was, with this focus on privacy, interested to see if the cheap sensors in normally available user devices could be used to follow a user's location as they move from a starting point. As the raw data from phone sensors was noisy and inaccurate the paper researched using a strapdown algorithm and applying different filters with different tuned values to get useful data for navigation.

They tested by applying a high pass filter, Kalman filter and some other filters that they tuned differently. They found several issues. These included gyroscope drift, noise, and large error rates even on the filtered data. Furthermore, the optimal filter configuration depended on the context of usage. If there was only acceleration for a short time, the high pass filter was preferred, but the Kalman filter performed better on longer acceleration. This is an issue for INS since it is preferred to have one filter that performs well in most conditions. The iPhone 3GS performed the worst due to not having a gyroscope available. This is despite the researchers also applying additional filters that were unneeded for the iPhone 4. There was too much interference with the magnetometer in the iPhone 3GS from other magnetic forces which gave it a poor performance in navigation. The paper concluded that even when applying filters it was challenging to create a useful navigation system because of the high error rate and noise in the data. With the iPhone 4 results were acceptable for a brief period but extended usage would deteriorate the quality of the result due to drift and error buildup.

Different from Corina Kim Schindhelm et al. [6], we (i) employ angle correction to compensate for the drift in the gyroscope; (ii) Incorporate step detection to calculate the distance a user has walked; (iii) Explore different ways of extracting and filtering data to use for drawing as explained in Section 4.5.

3. PRELIMINARIES

The following section will explain essential topics for the development of our application.

3.1 Inertial Measurement Unit and Navigation System

An Inertial Measurement Unit (IMU) combines the accelerometer, gyroscope, and sometimes the magnetometer. It measures linear acceleration and angular velocity. By measuring the change in velocity and the change in angular velocity, an IMU can give an estimation of the displacement of an object and in which direction it moved. The quality of the hardware in the sensors can vary. Good sensors will remain accurate for longer with less error build-up. IMU is classically used in inertial navigation systems for planes and cars to navigate. The distance and direction that has been traveled will be estimated by IMU to give a smooth UI showing movement and then GPS is used for correction to avoid error slowly building up over time. If a magnetometer is implemented in unison with the accelerometer and the gyroscope, it is classically used for correction of the drift in the gyroscope, but this is of less importance if using a GPS for error correction.

Research is also being done to replace GPS with IMU, since these do not rely on having a connection and they are also cheaper and more lightweight devices. The main challenge in this is error build-up in the accelerometer. The gyroscope drifts and builds up error too, but assuming the magnetometer is accurate this error can be corrected. Without using a GPS to correct position there is nothing in an IMU to correct for error build-up in the accelerometer [1]. The distance traveled that the accelerometer can accurately measure will vary depending on the hardware's quality but the error will build up over time regardless of sensor quality [7]. For an IMU to be able to replace GPS, it would require the accelerometer to have almost no error build-up. Alternatively, it would require creating a new sensor that could be used to correct the accelerometer, while also being cheaper to use than GPS. Section 3.3 explains the sensors that an IMU uses.

3.2 Step detection

Step detection classically consists of using the accelerometer to detect a user taking steps. Peak detection is mostly used for this. Here, each peak in the accelerometer sensor data is used to represent 1 step, and then each peak is counted to know how many steps have been taken. A trait they all have in common is that they generally require to be placed on a specific part of the body for the most accurate measurements. Otherwise, additional noise in the sensor readings might introduce inaccuracies in the amount of counted steps. Some step detection algorithms might also make use of the gyroscope and even the magnetometer for increased accuracy [9].

3.3 Sensors

This section will cover the sensors we make use of in our application. The quality of these sensors can vary but we are using the sensors integrated into mobile phones which entails they often are not of the highest quality.

3.3.1 Accelerometer.

The accelerometer is a sensor measures the acceleration of the object it is attached to. Accelerometers are typically used in inertial navigation in combination with other sensors to model the position and orientation of, for example, cars, aircraft, ships and more [8]. Modern phones usually have 3D-accelerometers installed in the device, where outputs of this system are given as (*x, y, z*) readings, as seen in Figure 1. An (*x, y, z*) accelerometer reading measures acceleration in the roll, pitch and yaw axes. The values on the x-axis are positive if the phone is rolled to the right, and negative if the phone is rolled to the left. Likewise, if the phone is moving forward, the values on the y-axis are positive, and if the phone is moving backward the values are negative. Lastly, if the phone is elevated, values on the z-axis are positive, and a downward motion will result in negative values.

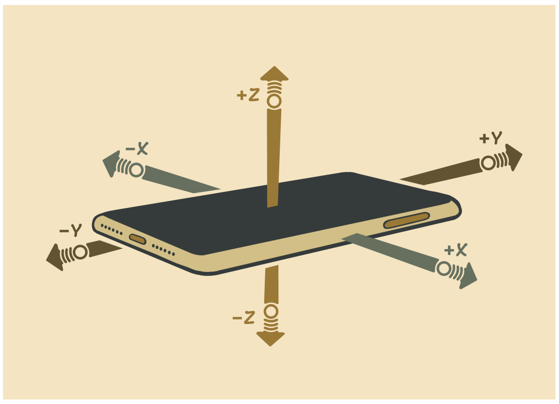

Fig. 1. XYZ-axis accelerometer [10].

The accelerometer can also be used to measure whether or not a user has rotated their mobile device, after which the mobile interface may be rotated [4]. A problem with the accelerometer, especially MEMS accelerometers which are used in phones, is that any error, for example constant bias on the readings accumulates over time. When readings from the accelerometer are used to calculate displacement over time, the constant bias on the readings of the accelerometer grows quadratically over time due to double integration performed on each accelerometer reading [8].

3.3.2 Gyroscope.

The gyroscope is a sensor that measures angular velocity. This can give knowledge about the current rotation. A problem with the gyroscope is that it drifts and accumulates errors over time. This causes the gyroscope to require to be resynced regularly [1]. Same as the accelerometer, the gyroscope gives output in the form of *(x, y, z)* values as seen in Figure 2. The parameter *x* works similar to a compass. If you place the phone on a flat surface and rotate it, *y* and *z* should be zero or close to zero and only the parameter *x* should change. It ranges from 0 to 360 representing a full circle of rotation. The *x* value will start at around 0 when starting the app then if rotated to the right, the value will increase and likewise, rotation to the left will cause the value to decrease until reaching 360 and 0 respectively. If increased beyond 360, the value will go back to 0 and if decreased beyond 0, it will increase to 360.

The parameters *y* and *z* values function similarly with their

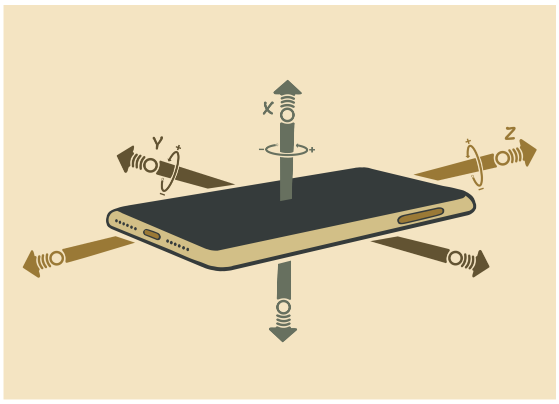

Fig. 2. XYZ-axis gyroscope [10].

respective values and rotations. The value of *x* can be used to see if a user turns. While the phone is on a flat surface, tilting the phone forward will cause *y* to increase and tilting it backward will cause *y* to decrease. The value of *y* is between 180 and -180 and is used to remove the gravity from the accelerometer. This is done to ensure gravity does not affect the final filtered data from the accelerometer [6]. The value of *z* will increase when tilting the device to the right and decrease when tilting the device to the left. The value of *z* can only be between 90 and -90. A consequence of this could be that we cannot know if a phone is tilted on its right side or left side. However, by tracking the z-value as it tilts, we can use these previous values to determine if we have tilted to the right or left side. Without doing this, we cannot differentiate if we are fully tilted to the right or left.

3.3.3 Magnetometer.

The magnetometer is a sensor that measures the direction and strength of surrounding magnetic fields. It is commonly used as a compass pointing north to the magnetic field. The main goal is to give us a reliable value we know to be correct. However, a problem with the magnetometer is nearby magnetic fields can cause interference. As mentioned the gyroscope tends to drift therefore a common usage of the magnetometer is to resync the gyroscope. This is done by comparing the value difference between the gyroscope and the magnetometer. For example, if a device is started that makes use of the magnetometer and gyroscope the initial difference in their values can be used to detect error build-up. If this initial difference changes we trust the magnetometer to be correct and adjust the gyroscope to restore the original difference. This allows us to correct the drift of the gyroscope [1].

4. METHODS

Algorithm 1 gives an overall overview of our entire application. The data from the accelerometer, gyroscope and magnetometer is extracted. Each time an event happens, which in this case is the user moving, this data is collected. This data is added to a buffer for each sensor. After extracting the data, preprocessing is done. The pre-processing mainly exploits the gyroscope y-axis value to remove the effect of gravity on the accelerometer. The accelerometer and gyroscope x-axis value are then used to calculate the distance walked. This calculation is returned as a vector consisting of x and y coordinates. This vector is added to a buffer and readings from this buffer are used to draw lines on a canvas, 4 times per second, as the user walks.

Algorithm 1 Overall overview of code

```
1: Input: Raw sensor data, $D$
2: Output: Canvas arrow drawings, $C$

3: // Data extraction
   $D_A$ $D_G$ $D_M$ $\leftarrow$ Extracting data from $D$, as explained in Section 5.2

4: // Filtering out gravity
   $filtered\_gravity\_D_A$ $\leftarrow$ Using filter that uses $D_G.y$ to remove the gravity from $D_A$ go to algorithm 2 for more details

5: // Calculate distance walked
   $V \leftarrow$ Using $filtered\_gravity\_D_A$ and $D_G.x$ to calculate the distance walked and then return it as a vector $V$ as explained in algorithm 3

6: // Add vector to array of vectors
   $V' \leftarrow V$ is combined with the previous value of array $V'$

7: // Draw route user is walking on canvas
   $C \leftarrow V'$ is used to draw the route the user is walking on a canvas.
```

4.1 Gravity compensation

As phones have a 3-dimensional accelerometer setup, as explained in Section *3.3.1*, gravity has an impact on the values in each axis of the 3D accelerometer. In order to compensate for this impact, we designed Algorithm 2:

Algorithm 2 Algorithm for gravity compensation

```
1: Input: A reading from the accelerometer, $A = (A_x, A_y, A_z)$
2: Output: A modified reading, $A' = (A'_x, A'_y, A'_z)$

3: // Extract values from the given reading $A$, explained further in Section 4.3.1
   $A_x$ $A_y$ $A_z$ $\leftarrow$ Initializing and storing values for each axis in the reading $A$

4: // Gravity compensation on the value of each axis
   $A' \leftarrow$ For each axis in $A$, perform the gravity compensation formula for that axis. Go to Section4.3.1 for more details
```

4.2 Calculating distance of user

One of the main goals of our application is that we can measure the distance traveled by the user of the application over a time span. In order to accomplish this goal, we have implemented an algorithm that uses accelerometer data to detect steps, or in other words *peaks* in the data, within a given buffer, and gyroscope data to depict the direction of the step. With this, in combination with a calibration of the step length performed by the user, we are able to measure the distance traveled by the user with little to no divergence relative to the actual distance traveled. The algorithm for calculating the displacement of a user over a time period is elaborated in algorithm 3.

Algorithm 3 Algorithm for calculating distance

```
1: Input: An array containing an array of accelerations $A = [(A_x, A_y, A_z)]$, and an array of gyroscope readings $G = [(G_x, G_y, G_z)]$
2: Output: A vector $V = (D, g_x)$, consisting of a distance traveled in the time step $D$, and a reading from the gyroscope $(G_x, G_y, G_z)$

3: // Select the maximum acceleration from the acceleration array, and the first gyroscope reading
   $a$ $g_x$ $\leftarrow$ Maximum acceleration $a$ and first gyroscope reading on the x-axis $g_x$

4: // Perform checks on the acceleration to determine if the acceleration is above or below the threshold to count it as a step
   $D \leftarrow$ Calculate distance using acceleration and the time difference between the current and the previous element in the buffer

5: $V \leftarrow$ Initialize the vector $V$ with the calculated distance $D$ and the gyroscope x-axis reading $g_x$
```

4.3 Preprocessing

4.3.1 Remove gravity filter.

Different filters are applied after extracting the raw data to remove noise from the data and thus make the data more meaningful. The gyroscope y-value is used to remove the effect of gravity on the accelerometer data as this would otherwise create noise in our data. We have a function that removes gravity, which takes as input one reading from the accelerometer and the gyroscope.

The arrays containing the accelerometer and gyroscope readings are implicitly expected to be of the same length. The function that removes gravity from the accelerometer readings is applied in a loop, which iterates over all accelerometer readings. To elaborate, this function takes 1 value from the accelerometer and the gyroscope and then uses the gyroscope value to remove the gravity from the accelerometer reading. After the loop has iterated over all accelerometer values, the function returns the accelerometer values, which have been compensated for gravity. The main function then pushes this value to a new list and the loop repeats. Then once the loop finishes the gravity will have been removed from all current accelerometer values, and an array containing these values will be returned. It should be noted that when the accelerometer values are from IOS, the values are inverted. In order to compensate the values accordingly, they are multiplied by -1 to convert them. Different calculations are done for each axis, which is the *x*-, *y*- and *z*-axes, listed in Equation 1, Equation 2 and Equation 3, respectively. The symbol $\pm$ is used, as the calculation differs depending whether or not the value on the given axis is below 0 or above 0. If the value is above 0, $-$ is used at the respective place of $\pm$, and $+$ is used if the value is below 0. This is the case for each axis, that is the formulas in Equation 1, Equation 2 and Equation 3.

$$acc'_x = -1 \times acc_x \pm (9.82 \times \cos(|gyro_y|) \times \sin(|gyro_z|)) \tag{1}$$

$$acc'_y = -1 \times acc_y \pm (9.82 \times \sin(|gyro_y|) \tag{2}$$

$$acc'_z = -1 \times acc_z \pm (9.82 \times \cos(|gyro_y|)) \times \cos(|gyro_z|) \tag{3}$$

4.4 Angle correction

After experimenting with the performance of the magnetometer and the gyroscope with regards to the correctness of the heading of the moving device, we decided to use the gyroscope for the purpose of monitoring direction change. There is a drawback from choosing the gyroscope as the sensor to measure the heading of the device, which is drift in the readings. In order to mitigate the drift in the readings from the gyroscope, we implemented the following formula to mitigate the error in the readings from the gyroscope:

$$\phi' = \lfloor \frac{\phi}{\mathbf{15}} \rfloor \times \mathbf{15} \tag{4}$$

where ϕ' is the corrected angle, and ϕ is the original angle. The magnetometer is typically used to correct the gyroscope drift as described in Section *3.3.3*. However, because of poor sensor quality, the magnetometer often seemed to be affected by interference when we were testing the application. For example, when heading north, the value was correct, but when the user was heading south it would not be the exact opposite of north. This gave some inaccuracies in canvas drawing. Due to these reasons, we implemented our own angle correction formula.

4.5 Step detection

In our first approach to a step detection algorithm, we performed extensive trial and error testing using accelerometer- and gyroscope data. This was done by applying different filters to evaluate how the filters affected the performance of the step detection. The performance of each filter was evaluated by logging calculations. These calculations were performed on the data in order to calculate the length the user had walked. These calculations were used to draw the route, which the user had walked, on the canvas. For short and medium distances, the calculated distance was generally reasonably accurate. However, the calculated distance for longer distances proved difficult to get an acceptable result, as the algorithm identified false positives as steps, as the length of the route increased. Due to the amount of steps detected being far from the actual step count by a vast amount, this resulted in a skewed drawing of the route and a large deviation in the calculated distance relative to the actual distance.

In order to solve the issue of not detecting all steps, we changed the logic of our algorithm. The reworked algorithm only extracts data we are interested in from the raw accelerometer data. For instance, the algorithm removes all negative data values, and therefore only performs calculations on positive acceleration data. This also means that we only focus on the different peaks above 0 isolated. The new logic defines peaks as positive values in a row until the values become negative again. The new logic also defines that each individual peak only counts if the peak contains more than 7 or more positive data points. This logic is implemented to filter noise from sightly shaking the phone, meaning that peaks that contain less than 7 data points are ignored. Once a peak has been detected each individual peak is used for calculating the user has walked. The logic of this algorithm also counts a step if a peak has 7 data points or more and it has an acceleration peak higher than 1.5 $\frac{m}{s^2}$.

We implemented the new algorithm to (i) calculate distance using only the accelerometer, (ii) calculate the distance using step detection using a fixed size step length of 0.75 meters per step and (iii) calculate a distance using a variable step length. The variable step length is recalculated each 5 steps taken by the user. The formula used to calculate the variable step length can be found in Equation 5, where the distance is the distance traveled between each 5 steps taken, the distance is calculated using the accelerometer data.

$$step\_length = \frac{distance}{steps} \tag{5}$$

5. EXPERIMENTS

In this section, we will describe how we have implemented the application. All functionality within the application is handled client-side, meaning there is no server for the client to communicate with, and all calculations are done locally in the app. It does not use an internet connection, which allows the application to be used in areas with bad or no connection.

5.1 Frontend

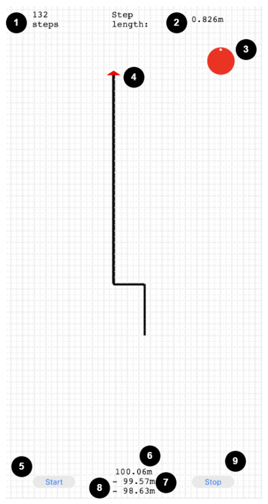

Fig. 3. The front end of our application, where (1) refers to step count, (2) refers to step length, (3) refers to the compass, (4) refers to user and trace, (5) refers to start sensors, (6) refers to distance, (7) refers to distance (varying step length), (8) refers to distance (fixed step length) and (9) refers to stop sensors.

Our frontend shown at Figure 3 consists of 9 different components. The red circle at the top right corner (3) serves as a compass and it can be in green or red color, where the user should aim for it to be green for more accurate results. When the compass is green, this indicates that the device is level, and vice versa. The 2D canvas starts with an arrow that represents the placement and direction of the user (4) and as the user walks the line is drawn. The trace drawing will begin when pressing start (5) and end when pressing stop (9). While the user is walking, the number of steps (1) and step length (2) can be seen on top. The distance is placed below, there exist three fields; one for showing the calculated distance based on accelerometer readings (6), one for showing the distance based on variable step length (7) and one for showing the distance using a fixed step length (8). The map also supports multi finger gestures for common functions like dragging the map, pinching for zooming and twisting for rotating.

5.2 Data extraction

In order to extract readings from the device's accelerometer and gyroscope, we must check whether or not the operating system of the device is iOS or not. The reason for checking the operating system of the device is that permission is required from the user of the application to extract sensor readings if the device is on IOS. Afterwards, if permission was granted by the user or if no permission was required, 3 event listeners are added to the client-side. One event listener is used for each sensor. Each of these sensors is activated whenever a new reading occurs on their respective sensors. When the user wants to get a new set of readings, they can press the 'Start' button. Once the user is walking a route with the application, the user can press the 'Stop' button to stop capturing readings from their device. The data extracted from the sensors is used to continuously depict the route that the user has taken.

5.3 Testing Setup

Testing of our application mostly consisted of walking different routes inside and around Cassiopeia[1] of varying lengths. The tests were conducted using standard mobile devices commonly available to most users, where the built-in sensors described in Section 3.3 were used. The app can be accessed and downloaded through a browser. By pressing the start button, the recording of the sensor data begins and a line will be drawn in the walking direction assuming that the user holds the phone as shown in Figure 4. The user can stop recording by simply pressing the stop button. When held at the optimal angle, the compass will be green as mentioned in Section 5.1. If it is not held in the optimal angle, the circle will be red. However, assuming that the angle is close to the optimal angle, a user can still get accurate readings even with the circle red. We have three main approaches to test our application. The first approach is to see what our application estimates the distance walked to be, compared to the actual distance. Google Maps can be used to give us a close approximate of the actual distance of these routes.

The second approach is to verify the amount of actual steps taken matches how many steps the application thinks have been taken. The third and final approach is to check that the canvas drawing matches the route on Google Maps. For instance, if the user has been walking straight in the same direction, we would expect the app to represent this by having drawn a straight line.

[1]Selma Lagerløfs Vej 300, 9220 Aalborg

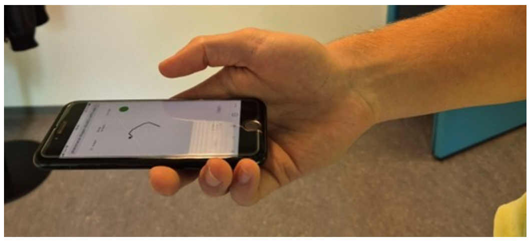

Fig. 4. Optimal angle to hold phone

If the user walks for a bit and then turns to the right, we would expect this to be represented on the canvas by a short straight line, a turn of 90 degrees and then continue drawing forward in this new direction. On longer routes, we compare by placing the drawn route on top of the actual map taken from Google Maps, as shown in Figure 9. As a result of this, we can see if the drawn route seems to match with how the route was walked.

5.4 Routes

This section covers the routes we walked to evaluate the performance and accuracy of our application. This includes short routes in a straight line, and short routes where the user changes direction while walking, and longer routes to evaluate the application in more realistic environments. The routes are presented in pictures, where the map behind is an image taken from Google Maps, and each line drawn on the map is a screenshot of the route drawn on the canvas in our developed application.

5.4.1 *Straight line.*

This route acts as a simple base case, where the user walks 14 meters in a straight line. This route is measured using a measuring tape outside our group room in Cassiopeia on AAU. This route can be found in Figure 5.

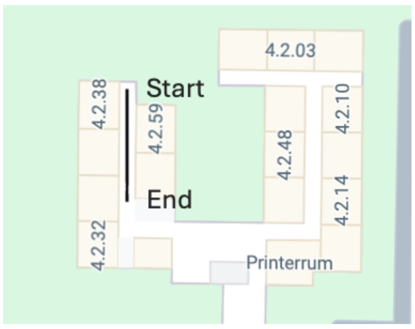

Fig. 5. A 14 meter route walked outside our group room in Cassiopeia.

5.4.2 *Left turn.*

This route is a bit more complicated compared to our base case route of 14 meters, as this route is 35 meters and contains a left turn. The route is also recorded outside of the group room in Cassiopeia. The route can be found in Figure 6.

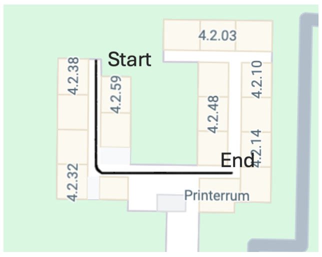

Fig. 6. A 35 meter route outside our group room in Cassiopeia.

5.4.3 *Straight with turns.*

This route is 100 meters long and spans across Cassiopeia. The route is walked on the top floor of Cassiopeia. The route begins outside our group room and ends at the opposite end of Cassiopeia on the walkway on the top floor. The route contains two 90 degrees turns, a left followed by a right turn, this route can be found on Figure 7.

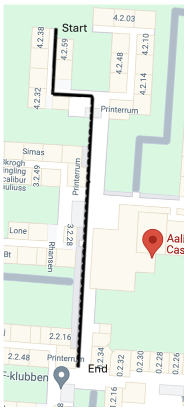

Fig. 7. A 100 meter route walked inside of Cassiopeia

5.4.4 *Straight long.*

This route is 200 meters straight. The route contains no turns or bends, and simply follows the bicycle road close to Cassiopeia, as this road is straight and long. We measured 200 meters of this road to be from the end of the bicycle shacks, to the middle of the T-junction on the bicycle road. This route is placed outside and not inside of Cassiopeia and can be found on Figure 8.

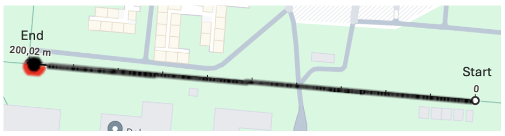

Fig. 8. A 200 meter route walked on the bicycle road next to Cassiopeia

5.4.5 *Inside Cassiopeia.*

Figure 9 shows a route around the second floor of Cassiopeia that was selected as an experiment. The total distance of the route is, according to Google Maps, approximately 233 meters. The route is one of the longest routes out of all of the experiments that we conducted. The reason for choosing this route was to compare our application's accuracy in both direction changes, but also the estimation of the distance when the route recorded by the user was longer.

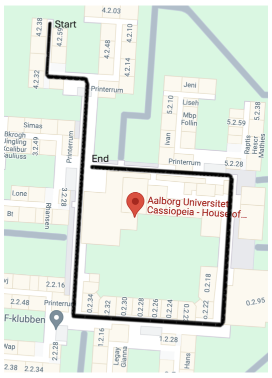

Fig. 9. A 233 meter route walked inside of Cassiopeia

5.4.6 *Around Cassiopeia.*

The last and longest route, which can be found on Figure 10, we selected to test the performance of our developed application. The route is an 800 meter route around the outside of Cassiopeia. The route begins from the main entrance of Cassiopeia, where the route begins by walking south to the bicycle road. The route then follows the bicycle road west towards Novi Science Park[2]. Afterwards, the route takes a right turn towards north and continues along the Novi Science Park building. The route continues to follow the parking lot at Novi Science Park and comes back on the opposite side of Cassiopeia, compared to the beginning of the route. The last segment of the route follows the northern path around Cassiopeia and ends at the same place as the route began.

The route has some elevation on a few segments. On the western end of the bicycle road (the road south of Cassiopeia) there is a negative elevation gain, this continues towards the north western corner of Novi Science Park building. The elevation is gained again on the northern segment around Cassiopeia, where the route gains elevation on the long segment from the last 90 degree turn until tile end of the route.

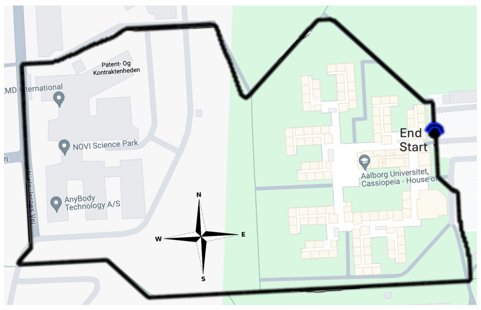
Patent- Og Kontraktenheden
NOVI Science Park
AnyBody Technology A/S
N
W
E
S
End
Start
Aalborg Universitet, Cassiopeia - House of...

Fig. 10. A 800 meter route walked around the outside of Cassiopeia.

[2]Niels Jernes Vej 10, 9220 Aalborg Øst, Denmark

Routes	Length	Steps	Length acc		Length step		Length fix step		Calc step	
			Meters off	%	Meters off	%	Meters off	%	Steps off	%
Straight	14m	18	0.3m	2.143%	0.2m	1.429%	-0.5m	-3.571%	0	0%
Left turn	35m	48	-0.4m	-1.143%	-0.6m	-1.714%	2.87m	6.086%	1	2.04%
Straight w. turns	100m	132	0.06m	0.06%	-0,43m	-0.43%	-1.37m	-1.37%	0	0%
Straight long	200m	240	1.3m	0.65%	0.02m	-0.01%	-19.25m	-9.625%	1	0.417%
Inside Cass	233m	310	1.42m	0.609%	-0.11m	-0.047%	-1,12m	-0.481%	1	0.323%
Around Cass	800m	986	17.58m	2,198%	14.13m	1,766%	-61.75m	-7,719%	2	0,203%

Table 1: Comparison of estimated and actual walking distances for various routes, with two metrics used to measure the deviation in meters/steps and the percentage difference.

5.5 Results

This section presents the results from experiments performed using the application. The experiments are meant to measure the performance of the application.

Table 1 compares the estimated distance walked for each route with the actual distance for these routes. The performance of the application is compared using two metrics: meters/steps off and the percentage deviation from the actual distance.

The application outputs three results: (i) a length calculated using the accelerometer and gyroscope (Length acc); (ii) a length calculated using step detection (Length step), where the step length varies and is calculated every five steps based on the distance moved; and (iii) a length calculated using step detection with a fixed step size (Length fix step), set at 0.75m per step. This fixed length was determined through experiments to be close to the average step length. The last metric measured is the prediction of the amount of steps taken (Calc step), this measurement is performed to test the performance of the step prediction.

5.5.1 Length calculated using accelerometer and gyroscope.

The results calculated using the accelerometer and gyroscope perform exceptionally well overall. In the shorter routes such as Straight and Left turns, which are respectively 14 and 35 meters, the deviation is a little larger than for longer routes. For instance, the route Straight and Left Turn, where they deviate respectively 2.143% and -1.143%. This deviation is due to the user stopping their walk shortly after starting. Specifically, the beginning and end of a walk can give some unintentional acceleration.

For the route Straight, the calculated distance is 14.3 meters, which is very close to the actual result, but as the calculation is 30 cm off, it gives a deviation of approximately 2%. The route Left Turn is calculated to be 34.6 meters, which is 40 cm off. As a result of this, the deviation is negative, as the application calculates the distance to be approximately 1% smaller than the actual distance. However, if the route is longer, then the calculation of the distance is better. For instance, in the 100 meter route (Straight w. turns), the application only measures a deviation of 0.06%. For the 200 meter route, the application calculated 0,65% more than the actual distance, resulting in a distance of 201.3 meters being calculated. This means that on shorter routes we are closer to the actual distance, measured in meters, but not closer in percentage deviation.

During our experiments, we found that elevation and wind conditions also has an influence on the accuracy of the application. On longer routes, such as Around Cass, the application measures 17.58 meters off the actual result. We experienced through our test that some of these extra meters come from walking up an incline. When a user is walking uphill, they are prone to take smaller but harder steps, which the application interprets as longer steps. This is one of the reasons why longer routes with elevation are more difficult to capture.

5.5.2 Length calculated variable step length.

As can be seen from the experiments with variable step lengths, all calculated distances are within 2% of the actual distances. This offers evidence that the developed app achieves high accuracy. More of the calculated distances, for instance Straight w. turns, Straight long and Inside Cass deviate with under 1% and Straight long deviates with 0.01%, which is close to the actual result.

The calculations are based on the same calculations from Length acc, we find the deviation to be a little larger on shorter routes. For the route Straight, the calculation is only 20 cm off, giving a total length of 14.2 meters being calculated, which results in a deviation of 1.429%. The length calculated from the Left turn calculates a total length of 34.6 meters, which is 40 cm off the actual distance. However, as the distance is quite short, this results in a -1.714% deviation. Overall, the length calculated by the variable step size performs as well, as the length of each step is also calculated using the distance calculated using the accelerometer. The results are similar to results from Length acc.

The prediction of longer routes with elevation suffers from the same problem as Length acc, where the application interprets steps taken uphill as longer as they actually are and the wind conditions on the given day that the experiment was performed. However, the final result of 14.13 meters off on an 800 meter route is quite satisfying, as the percentage deviation is only 1.766%.

5.5.3 Length calculated fixed size steps.

The overall performance of the calculations using a fixed step size gives good results in some cases, whereas in other cases it deviates by a vast amount. For instance, the calculations in the routes Straight w. turns and Inside Cass are close to the actual length giving a deviation of 1,37% and 0,481%, respectively. In other cases, it gives a deviation of 9.625%. This is most likely due to the step sizes being fixed, which is rarely the case in the real world. Therefore, it can perform well, but also deviate a lot from the actual result.

We found that on longer routes, such as Around Cass on 800 meters, fixed size steps struggle to get results within a good margin of the actual distance. This is most likely due to the elevation gain on the route and the wind conditions on the given day.

5.5.4 Measurement of steps.

The measurement of steps in the application detects almost every actual step taken. For instance, in the route Straight and Straight w. turns the application captures all actual steps taken. In other routes, such as Left turn, Straight long and Inside Cass, the application captures 1 step more than there is actually taken, resulting in a deviation of, respectively, 2,04%, 0,417% and 0,323% from the actual number of steps. The issue of recording more steps than the actual number usually arises when a user starts or stops walking, according to our experiments.

Our step detection overall performs well, even on long routes such as Around Cass, the application measures 2 steps more than the actual amount of steps, giving a deviation of 0.203%, which is a really good result when almost walking 1000 steps.

5.6 Comparison

As mentioned in Section 2 [6] concludes that, using the iPhone 3GS for navigation which does not have a gyroscope is difficult due to magnetic interference messing with the magnometer.This matches our own experience with using the magnetometer and is also why we decided to create our angle correction algorithm as covered in Section 4.4. This algorithm handles drift in the gyroscope without relying on the magnetometer for drift correction. Furthermore, the iPhone 4 gave acceptable results for walking short distances but then the error build-up and drift became too significant. The final result that [6] reported for the iPhone 4 was walking a circle with a 1-meter radius. They showed that applying filters significantly improved the raw data; however, the results were still inaccurate, and the endpoint did not connect properly with the starting point.
They concluded that extended usage and longer routes would result in even worse results because of drift and error buildup.

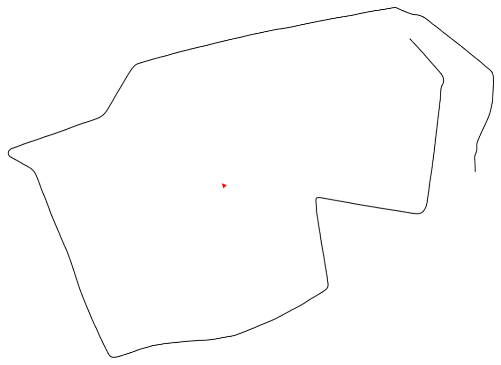
Fig. 11. First iteration of our app walking around Cassiopeia

Figure 11 is one of the first functional iterations of our app and made use of similar filter techniques as [6]. This drawing however was an 800m walk while their result was a circle with a 1 meter radius. We however see a similar end result where the starting point and ending point are not connecting properly and the drawing is inaccurate compared to the real route. Figure 10 shows this same route in the final version of our app. According to the map, This version of the drawing is closely identical to the actual route in particular the starting point and end point connect in Figure 10 which is one of the major challenges to achieve especially as the size of the route increases.

We thus conclude that our app has significantly improved accuracy compared to [6] thanks to the remove gravity filter, angle correction, data point filtering, and the algorithms we designed. Our design has better accuracy on short and especially long routes compared to [6]'s design, even though accuracy decreases as the route length increases.

6. CONCLUSION AND FUTURE WORK

We design and implement an inertial navigation system for navigating without the use of a GPS. We accomplished to do so by using only a phone's accelerometer, gyroscope and magnetometer. We have tested the accuracy on 6 different routes with various lengths and complexities and we can conclude that the distance measured using acceleration only, step detection with variable step length and step detection with fixed step length is within a reasonable distance from the actual length of the routes. Furthermore, we have implemented a frontend that enables the user to see their current location as well as enabling them to drag, zoom and rotate the map such that any part of the route can be visible can fit within the viewport. In this way, the application resembles the use of a map as a user expects a map to function.
Experimental results show that the proposed solution improves the accuracy of drawing maps and measuring distances compared to other existing solutions.

In the future, it will be of interest to (i) perform shake detection to omit shakes that are not from actual steps and (ii) display a 3D map that includes changes in altitude, orientation, and distance.

7. REFERENCES

[1] Norhafizan Ahmad, Raja Ariffin Raja Ghazilla, Nazirah M. Khairi, and Vijayabaskar Kasi. Reviews on various inertial measurement unit (imu) sensor applications. `https://www.ijsps.com/uploadfile/2013/1128/20131128022014877.pdf`, December 2013. (Accessed on 04/24/2024).

[2] Paolo Dabove, Giorgio Ghinamo, and Andrea LINGUA. (pdf) inertial sensors for smartphones navigation. `https://www.researchgate.net/publication/288842665_Inertial_sensors_for_smartphones_navigation`, December 2015. (Accessed on 05/20/2024).

[3] Google. Use navigation in google maps - android - google maps help. `https://support.google.com/maps/answer/3273406?hl=en&co=GENIE.Platform%3DAndroid`. (Accessed on 05/08/2024).

[4] George Grouios, Efthymios Ziagkas, Andreas Loukovitis, Konstantinos Chatzinikolaou, and Eirini Koidou. Sensors — free full-text — accelerometers in our pocket: Does smartphone accelerometer technology provide accurate data? `https://www.mdpi.com/1424-8220/23/1/192`, December 2022. (Accessed on 04/26/2024).

[5] Germán Rodríguez, Fernando E. Casado, Roberto Iglesias, Carlos V. Regueiro, and Adrián Nieto. Robust step counting for inertial navigation with mobile phones. `https://www.researchgate.net/publication/327756743_Robust_Step_Counting_for_Inertial_Navigation_with_Mobile_Phones`, 20 July 2018.

[6] Corina Kim Schindhelm, Florian Gschwandtner, and Michael Banholzer. Usability of apple iphones for inertial navigation systems — ieee conference publication — ieee xplore. `https://ieeexplore.ieee.org/document/6139701`, September 2011. (Accessed on 05/01/2024).

[7] Jinlong Song, Changde He, Renxin Wang, Chenyang Xue, and Wendong Zhang. A mathematical model of a piezo-resistive eight-beam three-axis accelerometer with simulation and experimental validation. `https://www.researchgate.net/publication/328546044_A_Mathematical_Model_of_a_Piezo-Resistive_Eight-Beam_Three-Axis_Accelerometer_with_Simulation_and_Experimental_Validation`, October 2018. (Accessed on 04/26/2024).

[8] Oliver J. Woodman. An introduction to inertial navigation. `https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf`, August 2007. (Accessed on 04/24/2024).

[9] Xiaokun Yang and Baoqi Huang. (pdf) an accurate step detection algorithm using unconstrained smartphones. `https://www.researchgate.net/publication/308834758_An_accurate_step_detection_algorithm_using_unconstrained_smartphones`, May 2015. (Accessed on 05/08/2024).

[10] Kaixuan Zhang, Dongbao Zhao, Linlin Feng, and Lianhai Cao. Cycling trajectory-based navigation independent of road network data support - scientific figure on researchgate. `https://www.researchgate.net/figure/Mobile-phone-three-axis-coordinate-system_fig1_352259249`, June 2021.

TETESTSETETSTE