Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing

SnowDroid – Your Private Snowboard Instructor

Bachelor's Thesis

Patrick Misteli

mistelip@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zurich

Supervisors:
Tobias Langner
Prof. Dr. Roger Wattenhofer

July 14, 2013

Acknowledgements

A very special thanks to my supervisor Tobias Langner for the initial idea and for supporting and guiding me through every step of this thesis from the beginning to the end.
In addition I would like to express my gratitude to Prof Wattenhofer who allowed me to investigate this interesting topic and complete my thesis on this subject.
Last but not least I would like to thank my parents and my friends Kevin Bocksrocker, Daniel Ackermann and Myriam Wirz for literally and/or figuratively supplying me with Ice Tea during the hard times.

Abstract

In this thesis the idea of detecting snowboard movements using standard phone sensors is investigated. When having an android phone attached to a person, the phone's sensors can give data about the current movement and position of the person. During this thesis an approach is stated that analyses these sensor readings in a way that allows detection of certain snowboard related movements. This can then be used to correct false snowboard riding techniques.
Using the methods and algorithms explained in this thesis it is possible to build an android app that allows detecting these errors and notify a rider by a simple press of a button. This can aid snowboard riders that have just recently started taking on the slopes.

Contents

Chapter 1

Introduction

1.1 Motivation

Living in a country mountains, snowboarding or skiing will most probably be part of one's winter activity. Most newcomers seek the help of a professional instructor to get a thorough introduction about the riding technique. After acquiring the basics the amateur rider usually takes off to ride on his or her own. While doing so it is very common that the rider will start developing incorrect movements due to natural instincts or by copying other riders on the slope. If uncorrected, these flawed riding techniques will limit the rider to reach an advanced level of skiing or snowboarding and furthermore endanger himself and other people sharing the slope with the rider. Taking further lessons from a professional is time and money consuming and most advanced amateurs will not see the need to engage in further lectures as they manage to ride a slope without falling over. Therefore a more practical solution is introduced in this thesis which allows an advanced amateur to have his or her mistakes pointed out by using their smart phones. The thesis is focused on snowboarding mistakes, but its methods could be adapted to be used for skiing as well.

1.2 Goal of Thesis

The goal of this thesis is to record movements of a snowboard rider with the use of an android cell phone and classify them as correct or incorrect riding techniques. The classification should be done with a useable and satisfying accuracy. It should allow an implementation of an android app that detects mistakes in real time and notify the rider thereof.

1.3 Outline

The thesis is organized in the following way:

Record Data
To be able to later write classifiers to separate the good from the bad, sensor data is collected with corresponding Video footage. This data includes correct and incorrect riding techniques.

Orientation of Phone
To make the sensor data independent of the phones orientation in the pocket, all phone-relative sensor data must be transformed into world coordinates.

Data Segmentation/CurveDetection
Once the sensor data is represented in world coordinates a single run is segmented into singular curves.

SVM
Using Support Vector Machine approach a classifier is generated to identify the property of a run. This classifier can then be used to tag new run segments

Chapter 2

Recording Training Data

To have training data for the classifier, the data must first be collected. To accomplish this, a rider was given multiple phones that he would place in different specified pockets. All phones would then start record their sensors simultaneously. Concurrently a second rider would record a corresponding video footage that can later be used to relate the recorded sensor data to real life events. This section is dedicated to how this was approached and executed.

2.1 Snowboard Layout

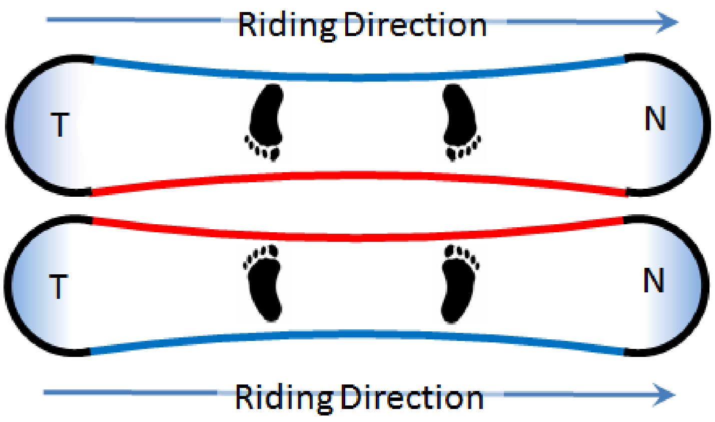


Figure 2.1: Snowboard Layout

Figure 2.1 shows the different part of a snowboard. The nose and the tail (Marked with “T” and “N”) of a snowboard is dependent on the riding direction. The **toeside (red line)** and the **heelside (blue line)** is dependent on the riders stance.

2.2 Type of errors

The following is a list of common mistakes advanced amateur snowboarders make. This list was generated by questioning various snowboarders deviating from amateur snowboarder to professional snowboard instructors.

Counter Rotation

The most mentioned mistake by advanced level snowboarders. This mistake happens when the rider is trying to terminate the curve too early. He does this by jerking around the snowboard in mid-curve. Not only does this look less elegant it is also dangerous when doing this at higher speeds. When the rider rotates the board with a lurch he has less control over it during that time. At lower speeds this is usually not a problem, but it prevents the rider from controlling his snowboard at higher speeds. Furthermore it hinders the rider to smoothly perform curves consecutively at higher speeds.

Leaning Backwards/Tailwards

Leaning backwards or tailwards is a mistake caused by an amateur's body instinct. Since the board is facing downhill the body is trying to keep away from the danger and therefore leaning in the opposite direction. Leaning tailside is dangerous due to various reasons. Since the *nose* of the snowboard is only slightly touching the snow the snowboard cannot be fully controlled and will not reliably respond to leaning to either side. With this, the snowboard is difficult to control even at lower speeds. In extreme cases the balancing point of the rider is shifted too much to the back of the board. This will result in the snowboard rotating by 180 degrees as the heaviest end of a snowboard will always point downhill.

Leaning Forward

Amateurs can tend to bend too much forward (leaning too much toeside). This is safer than leaning backwards (mentioned in the next point) but becomes inefficient. While being bent forward too much, a snowboarder is restricted in rotating his upper body which is essential for making a curve. Furthermore it is energy inefficient as leaning forward causes the knees to be more bent which increases the strain the quadriceps.

2.3 Recording Application

For the sensor data recording the app *DataRecorder* was programmed on android which is able to record all sensor activity and write its readings to a text file which can then later be processed.

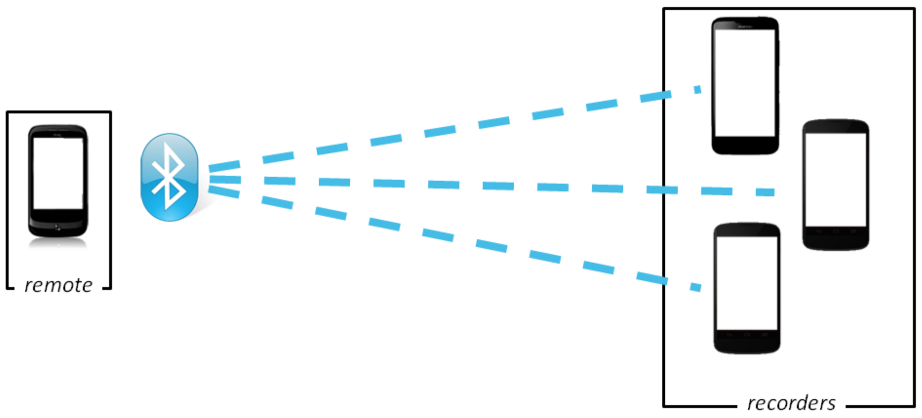


Figure 2.2: Communication Setup

2.3.1 Sensors

The priority of programming this app was to read as many raw sensor data and store them unadjusted on the phone's permanent storage. To ensure the availability of all possible readings the app stores all available sensors data (Accelerometer, Gyroscope, Magnetic Field and GPS Data). Additionally it stores the virtual Sensors (Gravity, Linear Acceleration and Rotation Vector) which are provided sensor fusion readings from the android operating system. Details about the different sensors are provided in section 3.1.

2.3.2 Usability

Device communication

Practical usage of the app was important. Most smart phones today (particularly those provided for the recording day) use a capacitive touch screen which is unresponsive when being touched by snow gloves. Therefore, next to the phones that are recording their sensor readings to their permanent storage (the *recorders*) an additional phone (the *remote*) was introduced which connects to the *recorders* via Bluetooth. The setup is shown in Figure 2.2 The *remote* is able to send a command to all connected devices to have them start or stop recording or simply return an "ACK" to confirm connectivity. Alongside the *start-recording* command the *remote* additionally sends a time stamp and a user defined run name. This information is used to tag the text file in which all *recorders* store their sensor data. Every command from the *remote* is answered by the *recorders* with an "ACK" to confirm the reception of the command. This way failure of the Bluetooth protocol is immediately observed

In case of permanent Bluetooth failure of a device the fail-safe app *DataRecorderMan* was programmed which is able to manually start the recording.

Bluetooth limits

The better hardware a device has the more Bluetooth connections can be maintained at once. Of the available devices the two flagships (HTC One X and Google Nexus 4) were able to connect to 5 Bluetooth devices simultaneously while lower end devices such as the Huawei Ascend D quad XL is only able to have two Bluetooth connections running at once. The *remote* would therefore ideally be a flagship to have as many devices recording at the same time. However, the flagship phones' sensors have a higher output frequency (details in Section 2.4.4) than the lower end phones. Additionally lower end phones have sensors that are not as sensitive as the sensors in the flagship phones. Using a flagship as a *remote* would inhibit the device from recording its own sensors data which would be a loss of readings from higher end sensors. A compromise had to be found between having one high end device connected to many lower end devices or a lower end device connected to a few high end devices. The optimised solution is elucidated below.

Services

The android operating system will terminate normal applications when being inactive for too long. This caused a problem, as the application should be able to be launched once and further on only be controlled by the *remote*. Standard implementations would cause the operating system to terminate the application, therefore android services had to be used. Android services are used for long-running operations that do not require user input. Furthermore the android operating system will not terminate a service even if the app that summoned it has not received user input over a longer time period.

2.3.3 Memory Management

The simple solution in recording sensor readings and storing it on the permanent storage of the device is to temporarily store the readings in the RAM and wait for the recording episode to be finished. Afterwards write the readings from the RAM onto the permanent storage. This works for shorter periods of time, however the device has to be able to steadily record a run for over 30min which would cause an "out of memory" error if this approach was used. This was solved by having multiple threads and queuing the readings. One thread is receiving the readings of the sensor and storing them in a public queue in the RAM. Another thread will continuously write all items of the queue to the permanent storage.

Once the device is told to stop recording the second thread will finish emptying the queue before terminating itself.

2.4 Recording Day

2.4.1 Location

As ski resorts were closing at the beginning of April due to ski season ending the choices of different locations were limited. The slopes that were needed had to be slightly and monotonously steep (difficulty *blue*). In addition the slopes had to be as smooth and as regular as possible. Riding down smooth and regular slopes allows focussing on executing the desired riding technique without having to adjust riding velocity and curve sizes due to small hills or change in slope steepness. This would ensure accurate data recording.

2.4.2 Time Period

All recordings were done on Monday 8^{th} of April 2013 from 09:30-15:30.

2.4.3 Skiing area & Slopes

The Ski resort Laax was chosen as a recording location. It features many different beginners slopes which are used by amateur snowboarders frequently. The slope that was chosen to make the recordings was the "Alp Dado". It is operated by a chairlift. This allows not only recharging the recording equipment on the way up but furthermore results in less time spent on the lift in comparison to a ski-lift which would have a slower operating speed.

2.4.4 Recording Equipment

The following is a list of recording equipment that was used on the recording day.

Camera

A Panasonic Lumix DMC-TS3 was used to capture the video footage. This model is able to record FullHD videos and is furthermore water- and freeze proof which allows outdoor video recording in snowy areas as required.

HTC One X

The HTC One X features a Panasonic Gyroscope sensor and a Panasonic accelerometer. Details about sensor readings are provided in section 3.1. The accelerometer sensors can deliver up to 61 readings a second. The gyroscope has a frequency of up to 216 readings per second *(*)*. Both frequencies are slightly lower when the android application is registered to multiple sensors at once. Due to the high frequency of the gyroscope this device was put in the riders chest pocket during the ride. This would allow an optimized capture of the upper body movement.
** Frequency of sensor readings were obtained by manually measuring them using a custom app*

Google Nexus 4

The Google Nexus 4 features a gyroscope and an accelerometer both manufactured by InvenSense. Both sensors can deliver up to 200 readings per second. To make use of the high frequency of the accelerometer this phone was put into the front pocket of the rider (the pocket pointing to the nose of the snowboard). This way movements and linear acceleration can be captured.

Huawei Ascend D quad XL

The gyroscope of the Huawei Ascend D quad XL can deliver up to 100 readings per second. Its accelerometer has an output frequency of 50 readings per second. This phone was used as an additional lower end data recorder for comparisons to the flagships.

HTC Desire

This device was used as a remote to activate and deactivate the other three devices. Its Bluetooth connection limit of three devices was therefore fully used. Its sensor frequencies are not listed here as this device did not record its sensor data.

2.4.5 Snowboarder Skill Level

A skilled snowboard instructor was recorded to ensure the error free curves are as flawless as possible. Furthermore an amateur rider who recently took up the sport was recorded to get realistic results on the errors. In addition the two riders use a different snowboarding stance. While the instructor is riding goofy (right foot forward) the amateur uses a regular stance (left foot forward). This should ensure that the classification of the curves are independent of the rider's stance.

2.4.6 Recording Day Layout

The aim was to capture all snowboard errors mentioned in section 2.2 with the sensors of the devices and record a corresponding video to the events. For comparison error free runs were captured as well. All run types were recorded for at least one entire slope run from the top to the bottom of the chairlift. The following recordings (including timestamps) were made:

- Recorded Instructor

  Error-Free run

  Fast Run

  Counter Rotation Run

  Leaning Forward Run

  Leaning Tailwards Run

- Recorded Amateur

  Error-Free Run

  Counter Rotation Run

  Leaning Forward Run

  Leaning Tailwards Run

2.4.7 Results

All types of runs were captured with the sensors and a corresponding video footage was recorded. A total of 10.4 GB of video footage was recorded and 1.12 GB of sensor data was obtained.

2.5 Data Visualizer Program

To visualize and closer examine the recorded data a Java program was created that would allow showing the selected data in a graph. Furthermore it should allow viewing of the corresponding video and draw a marker on the graph of the current position to allow synchronous viewing of the video footage and current recording of the data.

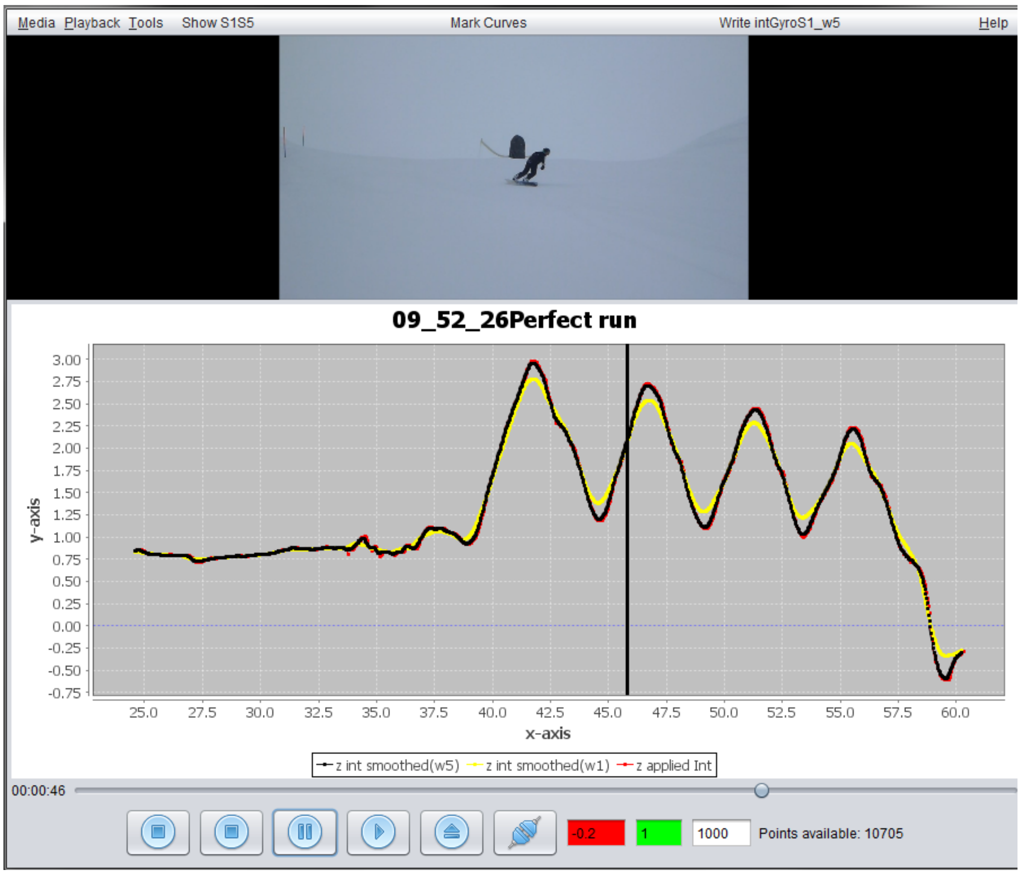


Figure 2.3: Data Visualizer

2.5.1 Video Player

The library vlcj (version 2.2.0) was used to create a java program that would play the video footage. The vlcj library allows embedding the VLC (VideoLan Client) player in a java program. The library features a function "getPosition" which returns the current position of the video. This information can be used to update the marker on the graph, displayed below the video, to point at the corresponding reading.

2.5.2 Graph

To display the graph the library jfreechart (version 1.0.14) was used. Jfreechart allows displaying line charts in a program. In addition it allows drawing a vertical marker on the graph. This was used to indicate the current position on the graph.

Chapter 3

Interpretation of Data

After having captured the different sensor data they need to be correctly interpreted and analysed. This section is dedicated to show how this was accomplished and how the raw data was used to detect a single curve.

3.1 Sensor Values and Coordinate Systems

3.1.1 Accelerometer

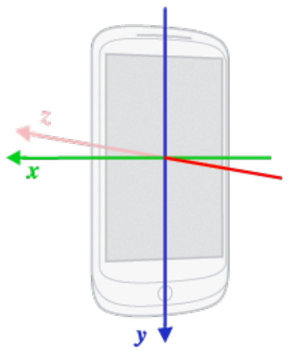


Figure 3.1: Accelerometer Coordinates $_{\text{adapted from}}$[4]

The accelerometer of a device will always show its current acceleration. Having the device at rest lying flat on a table will result in an accelerometer vector reading of

$$\begin{pmatrix} 0 \\ 0 \\ 9.81 \end{pmatrix} \tag{3.1}$$

This is an accurate estimation of gravity in a global coordinate system.

3.1.2 Gyroscope

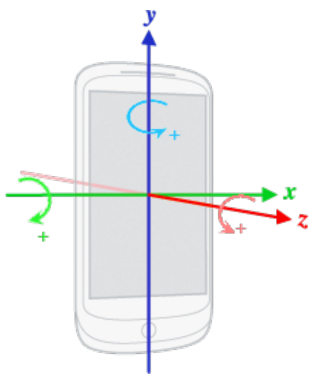


Figure 3.2: Gyroscope Coordinates adapted from[4]

A device's gyroscope will show its current angular velocity by returning a vector. The term "rotation vector" is used as the vector which determines the orientation of the rotation. To visualize this, one could imagine a stick pointing in the direction of the rotation vector and piercing through the middle of a device. If one were to rotate this stick with the device attached to it, the device would rotate in the direction of the rotation vector.
The values are always relative to the phones current position and do not give any information about the absolute global orientation of the phone. For e.g. when the device is flat on a table and rotated around its Z-axis the readings of the gyroscope will be the same as when the device is held flat against a wall and is rotated around its Z-axis. We must also keep in mind that the gyroscope outputs the rotational speed. This means that given an initial orientation of the phone (for example lying flat on a table) the rotational speed around each axis can be integrated over time. This results in three angles around the phones local axes (X,Y and Z shown in Figure 3.2) which indicate how far the device has rotated during this time period around each axis.

3.1.3 Magnetic Field

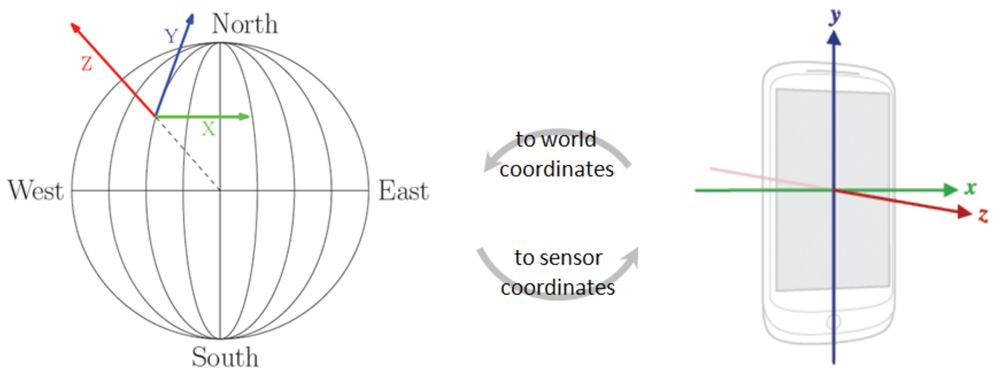
The magnetic field sensor returns the ambient geomagnetic field for all 3 physical axes. This sensor was not used due to various reasons. It is highly sensitive to electric noise, even to the device's own electric field when receiving a phone call. Furthermore the time needed to get an accurate new reading after a rotation is too slow for our use.

3.1.4 Justification for using both sensors

To clarify, the term “orientation” is used as the current rotational state of the phone in the world coordinate system (shown in Figure 3.3 (a)). This is independent of its current geographical position. For example putting the phone on a table face up and then face down results in the same geographical position, but the orientation of the phone in the world coordinate system differs
When the device is at rest determining the orientation is fairly simple, as we can simply take the accelerometer readings which will be a vector pointing downwards with a length of 9.81. This corresponds to the force of gravity $(9.81ms^{-2})$ acting upon it in the world coordinate system. When the device is accelerating in a direction this vector will bend into the direction of travel because in addition to the gravitational acceleration a second acceleration is present in the direction of travel. These vectors will sum up and the resulting vector will therefore not be pointing towards the centre of the earth anymore.
When the device is travelling with a constant velocity the accelerometer will once again point towards the earth. However, this scenario is also not applicable for use during a snowboard ride. The ski slopes are usually not perfectly smooth and a rider, especially an inexperienced one, is unable to absorb all bumps in the slopes with his legs. This will cause the accelerometer readings to have sudden jumps in its data making said readings on their own unusable for orientation purposes.
The gyroscope sensor is unaffected by linear acceleration of the device in any direction and will always output the rate of rotation around the phone’s local axes shown in Figure 3.2. This reading is limited to its own local coordinate system and does, on its own, not give any information about the current orientation of the phone in the world coordinate system. Furthermore integrating the values to get the corresponding angle will accumulate errors and the calculated angle would therefore drift from the actual angle. Compared to the accelerometer the gyroscope detects the new orientation faster after the device has been moved, as the accelerometer first needs to settle its new values again. In short, the accelerometer is able to show the new orientation of the device in world coordinates, with a small delay, if it is travelling at a constant velocity. The gyroscope is able to show the orientation in world coordinates faster, but needs to be given an initial orientation and needs to have its drift corrected over time.

3.2 Orientation of Device

To correctly interpret the gyroscope data we need to know the current orientation of the device relative to the earth’s surface. We fuse the data of the accelerometer and the gyroscope in order to have an accurate estimate of the device’s current orientation. *Pascal Bissig* created an app that is able to convert a vector from world coordinate to sensor coordinates and vice versa (Shown in



(a) world coordinates (b) device coordinates

Figure 3.3: Coordinate systems

Figure 3.3) using sensor fusion between the gyroscope and the accelerometer. He kindly granted permission to use his code in the experiments in this thesis. Using this sensor fusion enables conversion of any sensor data to world coordinates and therefore show not only the orientation of the device, but also the angular velocity of the device in world coordinates. When the sensor values of the gyroscope are transformed to world coordinates the third value (Z-value) will be the angular velocity around the Z-axis of the world coordinates. To further visualize this the following setup can be used.

As used as an example before, if the device is rotated flat on a table the gyroscope readings will show a rotation around the Z-axis. When converted to world coordinates the values will remain unchanged as the world coordinate system and the sensor coordinate system are currently aligned. Holding the phone against a wall and rotating it around its local Z-axis will result the gyroscope reading to once again show a rotation around its local Z-axis. However, if we rotate this reading to world coordinates the rotation vector will now point somewhere on the xy plane. The different coordinate systems are shown in Figure 3.3.

3.3 Curve Segmentation

After transforming the gyroscope readings to world coordinates an integration of the Z-axis will show the current rotation relative to the slope. Since the calculations should be independent of the geographic direction of the slope, no initial magnetometer reading was taken and an initial rotation of 0° was assumed. The next step is to separate the data into curve segments. A curve in rotation terms vaguely happens between the two points in time where the rider changes from transforming clockwise to counter clockwise and vice versa. This is shown

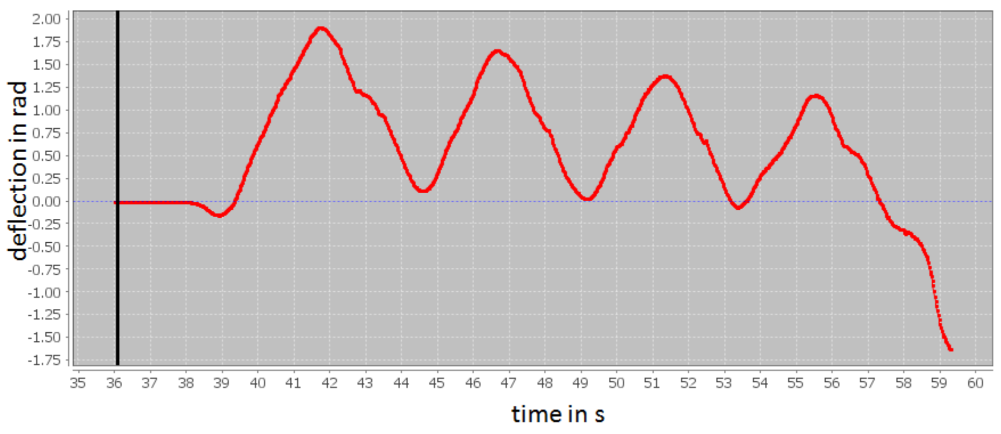


Figure 3.4: Perfect Curves, Z-Axis of gyroscope in world coordinates integrated

in Figure 3.4. Each extreme point marks the end of a curve and the beginning of the next one.

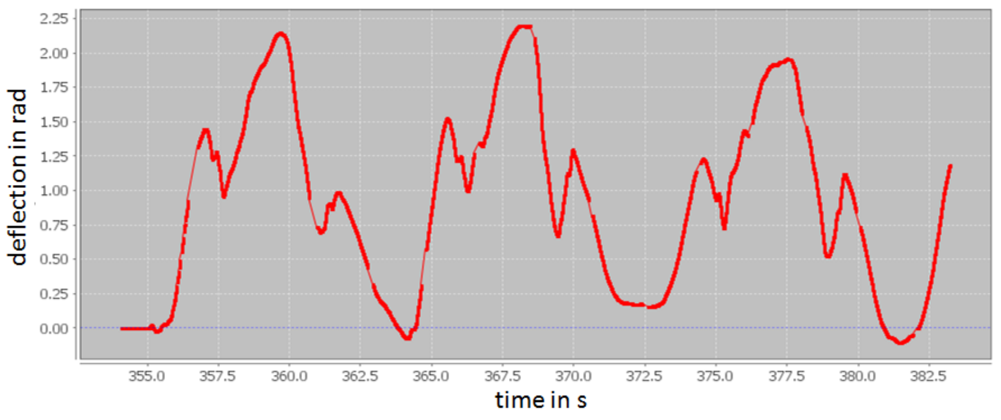


Figure 3.5: Counter rotation, Z-Axis of gyroscope in world coordinates integrated

However, this definition is too vague since the rider could have performed a counter rotation error during a curve. During a counter rotation the rider rotates his upper body in the opposite direction of the curve. The graph of the rotation of this is shown in Figure 3.5. A curve like this would cause the previous definition to falsely declare a curve as completed. A more robust curve detection algorithm must therefore be crafted, which will be elucidated in the following subsection.

3.3.1 Curve Detection Algorithm

Smoothing

In order to slightly diminish the ditch in the incline that a counter rotation error generates the data is smoothed by applying a weighted moving average over the data. The window of the moving average is set to 20 values with an exponential decay in weights. This means we take the newest calculated values with exponentially decaying weights and average them to the newest data item. The weight function is as follows:

$$f_w(i) = 1 \cdot (1 - 0.02)^i \tag{3.2}$$

Note that i is the number of the item in the current window starting from the item closest to the newest item. If we were to obtain the value of data item number 100 we take the reading from item 100 and set its weight to 1. We then take item number 99 of the already calculated average and set its weight to $f_w(1) = 0.98$. We continue iterating backwards until we have reached a windows size of 20. After multiplying each value with its corresponding weight we divide the sum of all items by the sum of their weight to gain the newest value. Furthermore this was extended to allow setting an custom weight("initial weight") for the newest value. The function for each value is therefore as follows:

$$new(k) = \frac{orig(k) \cdot initW) + \sum_{i=1}^{20} (new(k-i) \cdot f_w(i))}{initW + \sum_{i=1}^{20} f_w(i)} \tag{3.3}$$

for $k =$ [21,NumberOfItems in orig]
where $orig = serieswhichistobesmoothed$
$initW = initialweight$
for $k = [1, 20] : new(k) = orig(k)$

Using moving average will smooth over small ditches. We can however not rely on smoothing alone to smooth out all counter rotation ditches in the data. The more smoothing we apply the more the ditches will disappear. Applying too much smoothing will smooth out smaller curves and render them undetectable. Therefore a combination of smoothing and increase/decrease detection was chosen.

Increase/Decrease detection

Using the smoothed dataset we can more robustly detect the extreme points of the data. Most jitter in the data will have been ironed out by the smoothing. Simply looking for the next increase or decrease from point to point will however

not work as small jitter in the data still exists. This jitter is due to the high sensitivity of the sensors. To eliminate this jitter more smoothing would have to be applied, which would lead to loss of small curve segments as mentioned above. There exists a more robust maximum/minimum detection algorithm that allows jitter in the data. A visual explanation is shown below. The algorithm iterates through the list while remembering the maximum and the minimum value. If the difference between the current data item (data point) and one of the two is larger than a given threshold we mark the item as *currently_increasing* or *currently_decreasing* at that location. The initial threshold used for this thesis was $25 \cdot \left(\frac{\pi}{180}\right) \approx 0.436$. This is justified by defining a curve to have at least a 25° rotation (after smoothing). Furthermore this threshold is adjusted after each found curve segment. A rider's consecutive curves usually have a similar size. After finding a curve the threshold for detecting the new curve is set to $\frac{1}{3}$ of the previous curve. This means that a curve must be at least $\frac{1}{3}$ of the previous curve to be detected as such.
It must be taken into consideration that when a rider stops he/she may rotate by 360° before continuing to ride. There could also be a larger curve in the slope the rider is currently on. This would force the rider to do a curve that may be above 180°. This would set the threshold of a minimum curve to an unrealistic height. Therefore a maximum threshold of $25 \cdot \left(\frac{\pi}{180}\right) \approx 0.436$ is set.

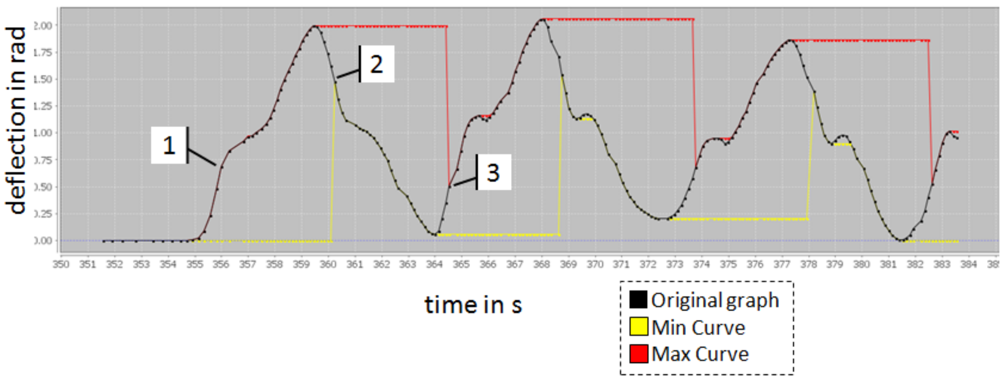


Figure 3.6: Curve Detection

3.3.2 Max/Min Detection

Using the increase/decrease algorithm elucidated before we can now search for maximum and minimum data points within the given increase/decrease segments. This will give a robust curve detection that allows counter rotations in between to happen.

3.3.3 Visual Representation of algorithm

Figure 3.6 demonstrates this algorithm in action. At marker **1** the distance between the current item and the minimum is larger than the given threshold (≈ 0.436). The algorithm therefore declares this data point as a new increase. We reset the maximum value, which will not have any effect at this point. At marker **2** a new decrease is registered the same way as the last increase was detected, only with the distance between the current point and the maximum this time. Since a previous increase was already detected we find the maximum point between $x_{increase}$ and $x_{decrease}$. We furthermore reset the minimum in order to correctly detect the next increase again. This happens at marker **3**. After finding the new local minimum between marker **2** and marker **3** an entire curve segment was detected. This now enables adjustment of the threshold. This is done by looking at the y-range between the last two extreme points and setting the threshold to $\frac{1}{3}$ of this. We continue to iterate through the entire run.

The setup now allows separation of a single run into curves. Additionally the information if it was a decrease or an increase in angle is stored in the curve. This indicates the direction of the curve which enables better classification later.

Chapter 4

Classifier

4.1 Labels

Using the curve detection algorithm presented in Section 3.3, the data was separated into curve segments. With the aid of the corresponding video each curve segment was manually labelled as one of the following:

NOCURVE (156 Samples)
Indicates an incorrectly segmented curve. It mostly occurs when the rider is resting for a longer period of time and then turns his upper body to continues to ride.

PER (227 Samples)
Indicates a (more or less) error free (perfect) curve

CROT(71 Samples)
Indicates that the rider has performed a counter rotation during this curve

LFOR (47 Samples)
Indicates that the rider was leaning too much forward during this curve

LTAIL (47 Samples)
Indicates that the rider was leaning too much tailwards during this curve

Total: 548 labeled runs

4.2 SVM (Support Vector Machine)

Support Vector Machine are learning models used for classification analysis. This corresponds to our problem as we attempt to classify each run to have a certain property (see Section 4.1 above). In the end a classifier takes a set of scalars from the run data and predicts the classification of the given run. Scalars can be calculations or direct values taken from the data, for example the total time span or the total angular span of the run after applying smoothing. To be able to do this the classifier first needs to be trained with already classified runs. This is done by feeding it with a set of scalars and the given class. Once the classifier is trained it can be used to predict unclassified runs by feeding it with the same set of scalars of the new run.

To accomplish this the *WEKA* [5] library was used. When using *WEKA* the data is made up of instances. One instance corresponds to a run and has different attributes. Attributes contain an attribute description and a scalar that was mentioned above. Once all instances and their corresponding attributes have been set *WEKA* can train a classifier.
Furthermore *WEKA* allows crossvalidation of data using a specified number of folds. The method to separate the data into a train set and a test set was not used due to the following reason. The train set may only be used to train the classifier while the test set is used to verify its accuracy. This results in a loss of training data which could have been used to further train the classifier. Using training data as test data will not produce accurate results as a classifier can simply match a test instance with an already trained instance. Crossvalidation separates the data into n sets (number of folds). *WEKA* then uses $n-1$ sets as training data and the remaining set as a validation set. It does this n times using a different set as test set every time. It then averages the performance of all folds.

WEKA has an extensive collection of classifiers. After research a subset of the classifiers were used and their results compared. For all properties the J48 classifier has shown to achieve the highest accuracy. J48 is an open source Java implementation of the *C4.5* algorithm [1] developed by *Ross Quinlan*. The *C4.5* algorithm generates a decision tree from training data and can be used to classify a range of labels (instead of a simple true/false classification which is performed in this thesis).

4.3 Chosen scalars

This section is dedicated to show the set of attributes that were chosen to classify each property of a run. An attribute name always consists out of the source data, the modification of this data and the value that was inspected of this modulated data. The attribute selection was done by running a forward and backward stepwise regression algorithm on an initial of 43 attributes. In a backward stepwise regression all 43 attributes are initially fed to the classifier while always removing one. This is done 43 times and the accuracy of the crossvalidation is stored. Taking the highest accuracy, backward stepwise regression is applied to the remaining 42 and so forth until no attributes are left. Forward stepwise regression is similar but the algorithm is starting with an empty attribute set and always adding the one attribute that increases the accuracy the most. The results of both algorithms were then compared and the set of attributes that delivers the highest accuracy was chosen. The reason why using all attributes does not result in the highest accuracy is as follows. Some attributes do not contribute to a better accuracy, but rather throw off the classification. For example if most of the correct curves happen to be 3 seconds long and the incorrect curves happen to be 4 seconds long the classifier will deduce that a longer curve is an incorrect curve. When testing this with all given runs the accuracy will be lower than if this attribute was not taken into account at all.

4.3.1 Data Sources and Modulations

The following data modulations were made. Since some modulations are reused by inspecting different scalars of it all modulations are stated here.

GyroX_Rot, GyroY_Rot and GyroZ_Rot
: The gyroscope vector was taken and then rotated to world coordinates. This results in a vector showing the angular velocity around all three axes (X,Y and Z) in the world coordinate system shown in Figure 3.3.

GyroZ_RotIntApp
: Starting with the rotated values (explained in the previous modulation) the integral of the Z values are taken. This is done by computing the area under the line from each point to its successor. Afterwards the sum of each consecutive integral is stored which results in a new dataset. This new dataset corresponds to a curve that shows the current rotation relative to the initial rotation of the curve. It is also the same data that was used in the curve detection algorithm.

GyroZ_RotIntAppS1W5
: Starting with the summed up integral (explained in the previous modulation) a moving average algorithm is applied once. The smoothing is coded as follows $S\alpha W\beta$. The α indicates the number of times the smoothing was applied while β indicates the weight of the initial item. Note that the window of the moving average remains at 20 items and the items excluding the initial always have an exponentially decaying weight with weight function $f_w(i) = 1 \cdot (1 - 0.02)^i$. This is the same weight function as the one used for the smoothing in the curve detection algorithm.

The following is a list of the run properties we are trying to detect and the values/ attributes selected to accomplish this. The attributes are sorted according to their rating starting with the highest. An attribute with a higher rating has more significance in defining the result than an attribute with a lower ranking. The rankings were obtained from the *WEKA* library.

4.3.2 NOCURVE Values

To further strengthen the curve detection algorithm the incorrectly detected curves were labelled as NOCURVE and an additional classifier was used to predict whether a segment is a curve. This would improve overall performance since segments that are not curves are not processed further to analyse their properties about riding technique.

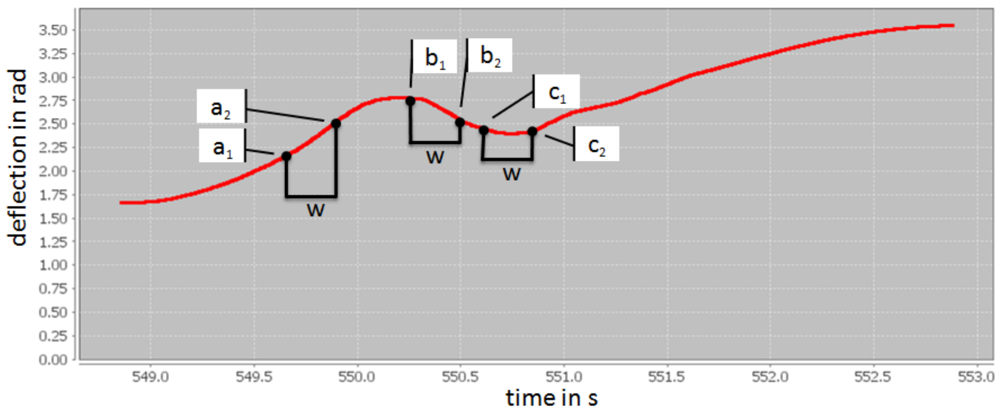


Figure 4.1: Gradients capture from a counter rotation curve segment

Total Time span

For disposal of falsely recognized curves the total time span was an important scalar. The total time span of a segment is the difference in time between the first and the last data's timestamp. The rider usually takes longer breaks than his time taken to do an average curve. Therefore longer time spans of a curve segment suggest that no curve was made in this segment. Since the time span is independent of what data we look at, this attribute name includes neither the source data nor the modulation thereof.

GyroZ_RotIntAppS1W5_PosGradCnt01

Taking the smoothed data a new array of data was created consisting of the gradients. The gradients are calculated by defining a certain time span as a window size. This window is then moved over the data while the gradient between the first and the last data point is stored. This is demonstrated in Figure 4.1 using a window of $w = 0.25s$. The window size for the current attribute is 0.1 seconds. The value of this attribute is equal to the number of positive gradients in this array. In Figure 4.1 a is a positive gradient while b is negative and c is zero. This number on its own does not say much. However, when the number of negative gradients is present in the classifier a ratio between the two is taken into account. Having mainly positive or negative gradients indicates a curve with a positive or negative rotation. Having equal amount of positive and negative gradients corresponds to the rider having not rotated at all. Note that this attribute can correlate with the PosNegGradSwitch attribute as well, which is elucidated later.

GyroZ_RotIntAppS1W5_MaxMinGradDif01
: As before, the array of gradients is inspected (window size 0.1 seconds). The scalar of this attribute is equal to the difference between the largest and the smallest (or most negative) gradient. A perfect curve will have a steady increase or decrease of angular rotation and would therefore result in a small number in this attribute. Having a higher number indicates uneven rotation around the world Z-axis.

GyroZ_RotIntAppS1W5_NegGradCnt03
: This is similar to the PosGradCnt01 attribute elucidated above. Here with a window of 0.3 seconds the number of negative gradients are obtained.

GyroZ_RotIntApp_TotalRadSpan
: This attribute is equal to the total range of the GyroZ_RotIntApp. This is equal to the total angle change of the rider during the curve. A curve starting from standing perfectly perpendicular to the slope looking downhill and ending at standing perfectly perpendicular to the slope looking uphill would result in a totalRadSpan of $180° \cdot \left(\frac{\pi}{180}\right) = \pi$

GyroZ_RotIntAppS1W5_PosNegGradSwitch03
: Again the gradient array is calculated with a window of 0.3 seconds. The value of this attribute is equal to the number of times the gradient switches from positive to negative and vice versa. A perfect curve will only be rotating in one direction and therefore have only negative or positive gradients. This results in PosNegGradSwitch to be equal to 0. The higher the number the more the rider has changed the rotation direction. This attribute can correlate with the number of either positive or negative gradients. In Figure 4.1 this attribute would have a value of 2 as it switches from positive to negative once and back to positive again.

Accuracy

Using the above attributes with the classifier J48 an accuracy of 97.992% was achieved. This is the percentage of correctly classified instances. From all runs labelled as NOCURVE 94.87% were correctly classified as such. Out of all other curves that were not labelled as NOCURVE 0.76% were incorrectly classified as not a curve.

Discussion

Using the total time span the classifier is able to take out most of the false positives. The other attributes contribute to the accuracy. If the rider rotates in many directions without having a dominant rotation it is a strong indication that no curve was made during this period. The curve segmenter and the chosen attributes therefore ensure a high accuracy in curve detection.

4.3.3 CROT Values

GyroZ_RotIntAppS1W5_MaxMinGradDif01
: Used in NOCURVE.

GyroZ_RotS1W5_MaxContra
: Since it is known whether a curve is an increase or a decrease of angular rotation (clockwise or counter clockwise) it is possible to detect counter movements that occur within the curve. If the curve starts at $0°$ and ends at $+180°$ the rider has ideally only applied rotations in the positive direction. This is visible in the gyroscope Z data after rotating it to world coordinates. If this scenario is the case this attribute stores the maximum negative value which corresponds to the maximum angular rotation in the opposite direction of the curve. Similarly this attribute stores the most positive value if the curve starts at $0°$ and ends at $-180°$. In Figure 4.1 a curve with an increase of angles was performed. Therefore for this curve segment the most negative angular velocity is stored in this attribute. Note that always the absolute value is taken to ensure independence of the curve direction.

GyroZ_RotIntApp_ContraCurveMovements01
: Using the algorithm used to detect increase/decrease in the curve detection algorithm, contra curve movements can be detected. The threshold in this attribute is set to 0.01 ($\approx 0.573°$). While iterating through all data points the minimum is stored. If the difference between the current data point and the minimum is above this threshold the "climb counter" is increased. The "drop counter" is increased similarly if the difference between the current item and the maximum is larger than the threshold. As with the previous attribute, it is known whether a curve is increasing or decreasing its rotation angle i.e. whether the curve was clockwise or counter clockwise. If the angles are increasing (such as in Figure 4.1) the ContraCurveMovements attribute stores the number of times the data decreased ("drop counter") during the curve. Analogically the "climb counter" is stored in this attribute if the curve mainly consists of a decrease in angles.

GyroZ_RotIntAppS1W5_PosNegGradSwitch03
: Used in NOCURVE.

GyroY_Rot_Max
: After the gyroscope values are transformed to world coordinates this attribute is equal to the maximum angular rotation around the Y-Axis.

GyroZ_RotIntAppS1W5_NegGradCnt03
: Used in NOCURVE.

GyroX_Rot_Min
: After the gyroscope values are transformed to world coordinates this attribute is equal to the smallest (or most negative) angular rotation around the X-Axis.

Total Radspan
: Used in NOCURVE.

GyroZ_Rot_Min
: After the gyroscope values are transformed to world coordinates this attribute is equal to the smallest (or most negative) angular rotation around the Z-Axis.

Accuracy

Using the J48 classifier an accuracy of 93.96% was reached. This means that from all runs 6.04% were incorrectly classified. Of all the counter rotations 84.51% were correctly classified as such and from all the correct runs 3% were incorrectly classified as a counter rotation

Discussion

The counter rotation can be detected rather well. This is due to the fact that the rider moves his upper body in the opposite direction of the curve for a short period of time. 100% accuracy is difficult to achieve since the counter movement varies of magnitude between each rider. The riders that are aware of this error and are trying to avoid it will usually have a smaller magnitude of counter rotation. This was especially visible in the data when the amateur rider was recording his curves with no errors where sometimes minimal counter rotations would occur nevertheless. These curve segments were then therefore not classified as PER. Riders that are unaware of the incorrect riding technique will perform a counter rotation that is more obvious to the eye and to the sensor since his upper body will perform a more obvious counter rotation.
Theoretically speaking the smallest rotation into the opposite direction of a curve could be declared as a counter rotation. Due to the unevenness of the slope the rider must absorb small hills with his body to ensure control of his snowboard.

While doing so avoiding even the smallest rotation in the opposite direction is nearly impossible. Therefore it was necessary to have a certain threshold of counter rotation and further take other attributes into account when deciding whether a counter rotation was performed.

4.3.4 LFOR Values

GyroY_Rot_Max
: After the gyroscope values are transformed to world coordinates this attribute is equal to the maximum angular rotation around the Y-Axis. (Same as used for detecting CROT).

GyroY_Rot_Min
: After the gyroscope values are transformed to world coordinates this attribute is equal to the smallest (or most negative) angular rotation around the Y-Axis.

GyroZ_RotIntAppS1W5_LongestStreak1
: Inspecting the summed up Z-integrals of the transformed gyroscope a vertical window (range window) is set with a certain threshold (here 0.1 which is $\approx 5.73°$). This attribute stores the longest time streak where the values are all within this threshold.

GyroZ_RotIntAppS1W5_ContraCurveMovements01
: This is similar to the*ContraCurveMovements* in the CROT detection with the difference, that the data is smoothed with initial weight 5 before running the algorithm on it. Threshold is at 0.01 ($\approx 0.573°$).

GyroZ_RotIntAppS1W5_PosNegGradSwitch02
: This is similar to the attribute used to detect NOCURVE with the threshold set to 0.2 seconds.

GyroZ_RotIntAppS1W5_PosNegGradSwitch03
: Used in NOCURVE.

GyroZ_RotIntAppS1W5_LongestStreak01
: LongestStreak attribute (explained above) with threshold of 0.01 ($\approx 0.573°$).

GyroZ_RotIntAppS1W5_PosNegGradSwitch01
: Similar to the previous PosNegGradSwitch attributes. In this occurrence the threshold was set to 0.1 seconds.

GyroZ_RotIntAppS1W5_MaxMinGradDif01
: Used in NOCURVE.

Accuracy

Using the J48 classifier an accuracy of 91.61% was reached. This means that from all runs 8.39% were incorrectly classified. Of all the runs that were declared as “leaning forward” 63.83% were correctly classified as such and from all the correct runs 2.64% were incorrectly classified as leaning forward.

Discussion

Leaning forward detection has a significant lower accuracy than counter rotation detection. This is due to the following fact. When a rider is executing a toeside curve he automatically leans into the curve to stabilize his posture. The faster his riding velocity and the smaller the curve the more the rider will lean into the curve. In other words he will lean forward even when he is making a correct curve. To the sensor in the chest pocket the two scenarios are difficult to distinguish as both indicate an upper body that is almost parallel to the ground.
To improve detection of leaning forward one could further process GPS readings to get the curve radius and the speed. Using this an optimal upper body lean angle could be calculated. This angle could then be compared with the actual leaning angle of the rider to detect whether he is leaning forward too much. It should be stated here, that this error is the most uncommon one from the three that were tested. It is furthermore the error that will limit and/or endanger the rider the least from the three when performed.

4.3.5 LTAIL Values

GyroZ_RotIntApp_ContraCurveMovements01
: Used in CROT.

GyroZ_Rot_Max
: After the gyroscope values are transformed to world coordinates this attribute is equal to the maximum angular rotation around the Z-Axis. (Similar to GyroY_Rot_Max used for detecting CROT).

GyroZ_RotIntAppS1W5_PosNegGradSwitch03
: Used in NOCURVE

GyroZ_RotIntAppS1W5_NegGradCnt03
: Used in NOCURVE

Accuracy

Using the J48 classifier out of all the runs that were declared as “leaning tailwards” 10.64% were correctly classified as such and from all the correct runs 0%

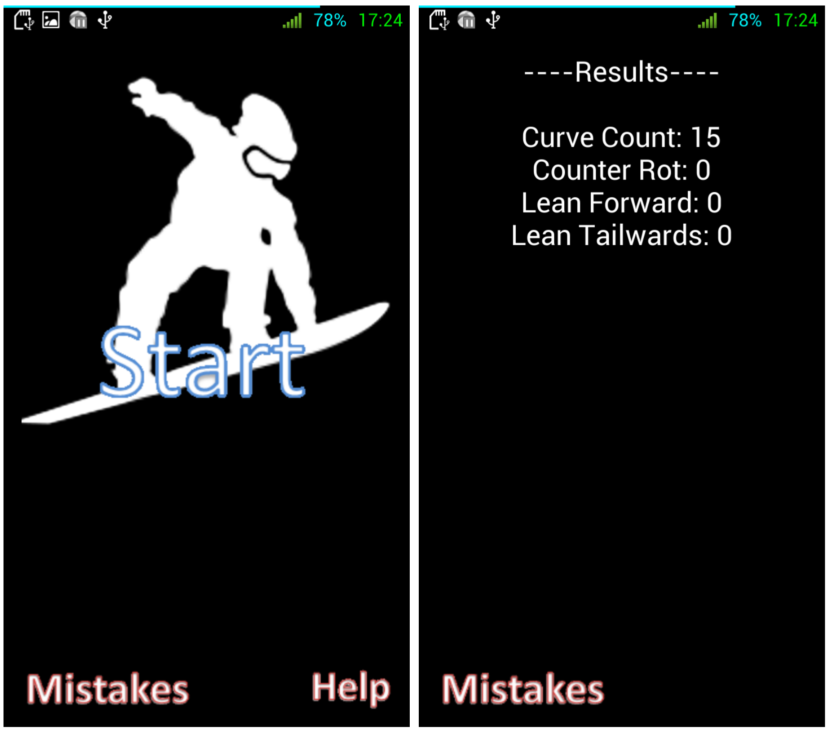
were incorrectly classified as leaning tailwards.

Discussion

This error is difficult to capture with a sensor. Nevertheless for the sake of completion this error was attempted to be classified as well. The error occurs when the rider is putting too much weight on the back foot. This does not guarantee, that his body is tilted tailwards i.e. the rider is able to put his weight on his back foot without changing his upper body pose. According to the sensors the rider is still standing straight on his board. To detect this error efficiently a weight pad could additionally be inserted onto the base plate with a Bluetooth communication to the phone. The base plate is the bottom part of the binding where a snowboarder stands on. These weight pads could register the weight distribution of the rider so he could be notified if his weight is mainly on his back foot.

Chapter 5

Android Application



----Results----

Curve Count: 15
Counter Rot: 0
Lean Forward: 0
Lean Tailwards: 0

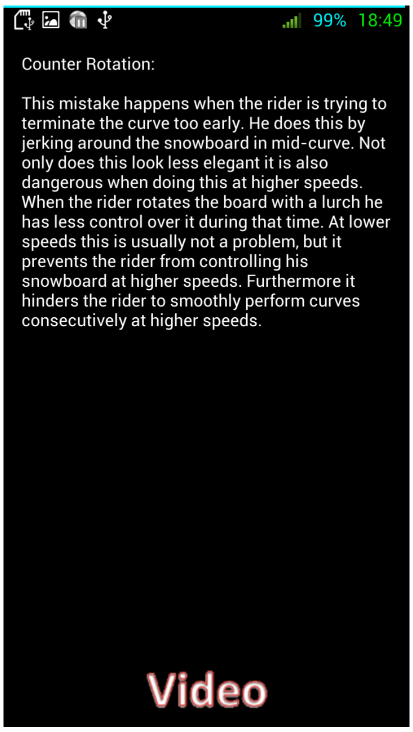
Figure 5.1: Snowdroid [3]

To show an approach of how the data and algorithms in this thesis can be applied to an android application a *Snowdroid* App was programmed. *WEKA* is able to store a trained classifier and load this classifier again to predict further segments. The original *WEKA* library does not run on android. Username *rjmarsan* has made a working version available online called *WEKAStripped* [2] that runs on android. Full functionality is however not guaranteed. The app should primarily help the user detect his riding mistakes and explain to him how to correct them. However, further additions can be made such as viewing one's

progress over an entire season or sharing one's high score (best error-free to error ratio on a specific slope) with friends via facebook or twitter.
The app is implemented in such a way that adding new classifiers and new mistake descriptions is simple. Adding more error detectors is therefore another extension point.

5.1 Usage

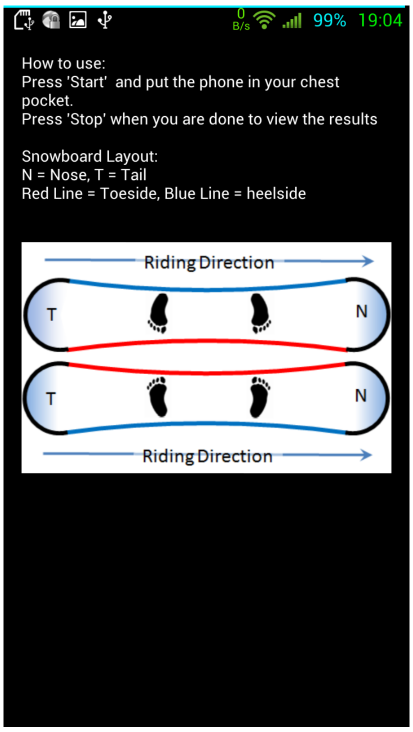
Figure 5.1 on the left hand side shows the initial screen when starting *Snowdroid*. After pressing start *Snowdroid* will start receiving the sensor values. The values are then segmented into single run segments using the method presented in section 3.3. From the segment the attributes that were presented section 4.3 are obtained and then sent to each classifier to predict its error property. When pressing stop the right screenshot in Figure 5.1 appears which displays the amount of curves made and the amount of mistakes made during those curves.



99% 18:49

Counter Rotation:

This mistake happens when the rider is trying to terminate the curve too early. He does this by jerking around the snowboard in mid-curve. Not only does this look less elegant it is also dangerous when doing this at higher speeds. When the rider rotates the board with a lurch he has less control over it during that time. At lower speeds this is usually not a problem, but it prevents the rider from controlling his snowboard at higher speeds. Furthermore it hinders the rider to smoothly perform curves consecutively at higher speeds.



0 B/s 99% 19:04

How to use:
Press 'Start' and put the phone in your chest pocket.
Press 'Stop' when you are done to view the results

Snowboard Layout:
N = Nose, T = Tail
Red Line = Toeside, Blue Line = heelside

Figure 5.2: Snowdroid

The user is able to browse through all different mistake descriptions and look at further details about each of them. This is shown in Figure 5.2 on the left hand side. In addition it is possible to view a video of the mistake to further visualize the incorrect riding technique. On the right hand side of Figure 5.2 the help screen is displayed. It shows how to use the app and displays the layout of the snowboard for better understanding of the error descriptions.

Chapter 6
Conclusion

After experimenting with sensor data and classifiers it can be stated that it is possible to detect snowboard post-amateur errors to a satisfying degree. Counter rotations can be identified with a satisfying accuracy. Since this is one of the most common errors beginner snowboarders perform the readings and classifier can be used to detect and notify the rider of this. Furthermore the curve detection algorithm can be used to not only count the number of curves the rider took, but also the consistency of curve size and curve time during an entire slope run. In our scenario multiple phones were used. Most people do not own more than one smart phone. Therefore the initial focus was to use only one device (in the chest pocket of the rider). Since this was accomplished with a satisfying accuracy the algorithms and technologies presented in this thesis can be used by a broader group of snowboarders. It should be mentioned that the detection of the mentioned mistakes does not replace the educated eye of a professional. A professional is able to point out mistakes sensors may be unable to capture. This also includes non-riding mistakes such as correctly tying the snowboard shoes and correctly mounting the snowboard which a phone is unable to detect.

Bibliography

[1] C4.5 Algorithm. http://www.rulequest.com/Personal/.

[2] WekaStripped by rjmarsan. https://github.com/rjmarsan/Weka-for-Android/.

[3] Background Image by Spreadshirt.net. http://www.spreadshirt.net/ruby-red-snowboarder-silhouette-women-s-t-shirts-C4408A14082628/.

[4] Android Developer. http://developer.android.com/reference/android/hardware/SensorEvent.html.

[5] Weka Library. http://www.cs.waikato.ac.nz/ml/weka/.