Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Kybernetics

Head Orientation Estimation Using Inertial Measurement Unit

Valášek Jiří

Supervisor: Ing. Smutný Vladimír, Ph.D.
May 2016

Acknowledgements

I am grateful to the supervisor Ing. Smutný Vladimír, Ph.D. . I am extremely thankful and indebted to him for sharing expertise, and sincere and valuable guidance and encouragement extended to me.

Declaration

I declare that I have worked out this thesis independently and mentioned all used information sources in accordance with the Guideline about observation of ethical principles while preparing college final thesis.

Prague, May 27, 2016

....
signature

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 27. května 2016

....
podpis

Abstract

The need for head tracking in indoor environment increases as augmented reality glasses and virtual reality devices are more applied. The approach is to use already built-in sensors in the devices. We use inertial measurement unit with 9 Degrees of Freedom which incorporates 3D gyroscope, 3D accelerometer and 3D magnetometer. We would like to test it in order to evaluate its suitability to track head orientation. The performance is demonstrated by drawing the symbol into live video image marking fixed direction in ambient space.

Keywords: augmented reality, virtual reality, inertial measurement unit, gyroscope, accelerometer, magnetometer, Android OS

Supervisor: Ing. Smutný Vladimír, Ph.D.

Abstrakt

Potřeba určování orientace hlavy vzrůstá s použitím brýlí rozšířené reality a zařízeními virtuální reality. Zvolený postup využívá v zařízeních již vestavěné senzory. Inerciální měřící jednotka o devíti stupních volnosti zahrnující 3D gyroskop, 3D akcelerometr a 3D magnetometr je nástrojem, které bych rád otestoval, abych stanovil jeho vhodnost pro dosažení daného cíle. Cílem je sledovat natočení hlavy v závislosti na čase a poté demonstrovat zjištěné výsledky. Demonstrace využívá kamery a rozšiřuje zaznamenané video o symbol určující směr v prostoru.

Klíčová slova: rozšířená realita, virtuální realita, inerciální měřící jednotka, gyroskop, akcelerometr, magnetometr, Android OS

Překlad názvu: Odhadování orientace hlavy za pomoci inerciální měřící jednotky

Contents

Figures

Tables

Chapter 1

Introduction

People searching for items in complex environment without global navigation satellite system such as warehouse, mall etc. needs to be navigated in order to increase productivity. It is necessary to know position and orientation of their head in order to give them correct instruction where to go. This information can be shown on smartphone or through augmented reality glasses, which are becoming more affordable for common customers nowadays.

Smartphones can be used together with virtual reality glasses as a VR device. VR applications also need head orientation to display correct context. For example 360° video player require head orientation in order to show only corresponding part of video.

The aim of this work is to determine head orientation for augmented or virtual reality usage. With focus on low power consumption, limited computing power and usual indoor usage, we cannot use techniques such as natural features navigation or GLONASS/GPS. We also will not use external device, which would made augmented reality glasses, virtual reality device heavy and uncomfortable. Therefore we decided to use low-cost low-power micro-electro-mechanical system inertial measurement units. We decided to test inertial measurement units present in smartphones, because we assumed that same consumer grade micro-electro-mechanical system inertial measurement units are used in augmented reality glasses, virtual reality devices and smartphones.

For our experiments we decided to use Android smartphones. One reason is that regarding to research [1] Android Market Share for the fourth quarter of 2015 was 80.7 %. That is greater than iOS with 17.7 %, not to mention Windows with 1.1 % and the others with 0.4 %. Another reason is that Android is the most used operating system (OS) in augmented reality glasses e.g. EPSON Moverio BT-200, Google Glass, Optivent ORA 1.

Chapter 2

Inertial Measurement Unit

Inertial measurement unit (IMU) consists of three-axis gyroscope, three-axis accelerometer, three axis magnetometer. Modern micro-electro-mechanical system (MEMS) based inertial measurement units also incorporate thermometer to compensate temperature effects on measured values.

2.1 Measured quantities

IMU with 9 degrees of freedom (DoF) provides us 9 values, 3 numbers per each sensor type. IMU does not has to be single chip, but can be created by connection of accelerometer unit, gyroscope unit and magnetometer unit instead. Therefore accelerometer/gyroscope/magnetometer unit may have reference frame (RF) with different orientation in space than IMU RF. Thus, the set of 9 values from IMU consists of 3 values from each incorporated sensor in its RF transformed to IMU RF[1].

An accelerometer measures linear acceleration with respect to an inertial RF. Therefore, if an accelerometer unit has three accelerometers aligned with axes x, y and z of its RF, then each measured value is a linear acceleration of its RF with respect to an inertial RF expressed in accelerometer's RF.

A gyroscope can measure angular rates with respect to an inertial RF or rotation angle with respect to a specific RF. With an angular rates measuring gyroscope aligned to its RF axes, given 3 values correspond with a angular rates of its RF with respect to an inertial RF expressed in gyroscope's RF. However, with angle measuring gyroscope aligned to its RF axes, given 3 values means rotation of its RF with respect to a specific RF expressed in the specific RF. The specific RF could be for instance gyroscope RF in moment just after switching on.

A magnetometer measures strength of magnetic field in given direction. A magnetometer unit uses three magnetometers to quantify magnetic field

[1]If IMU uses angle measuring gyroscope then the 3 values are always expressed in a specific RF

strength in direction of its RF axes.

2.2 Description

IMU sensors use physical principles to measure quantities described in previous section 2.1.

2.2.1 Gyroscopes

In past, mechanical gyroscopes had been used as part of every gyroscopic flight instrument until they began to be replaced by more accurate ring laser gyroscopes (RLG) and less expensive micro-electro-mechanical systems (MEMS) in 1990s [15]. Gyroscopes can be based on several physical principles, which influence sensor features such as dimensions, precision, bandwidth.

However, only small low-cost low-power MEMS sensors are suitable for usage in a smartphone or another devices where weight and power consumption is essential. MEMS gyroscopes can be so small because they do not use bearings. They use vibrating parts only so they are also called micromachined vibratory gyroscopes (MVG). MVG are based on Coriolis effect and they can be classified into two basic categories - angle or angular rate measuring systems [22].

Majority of MVG are of angular rate measuring type, which uses different driven and sensed oscillation mode i.e. angular and linear mode. Tuning-fork gyroscope shown in figure 2.1 is example of angular rate measuring type.

Tuning-fork with 2 DoF consists of two tines with driven flexural oscillation. Oscillating tines are moving apart in first half-period (full red arrow) and getting closer in second half-period (dotted red arrow). When tuning-fork gyroscope is subject to angular rate (green arrow), Cori-

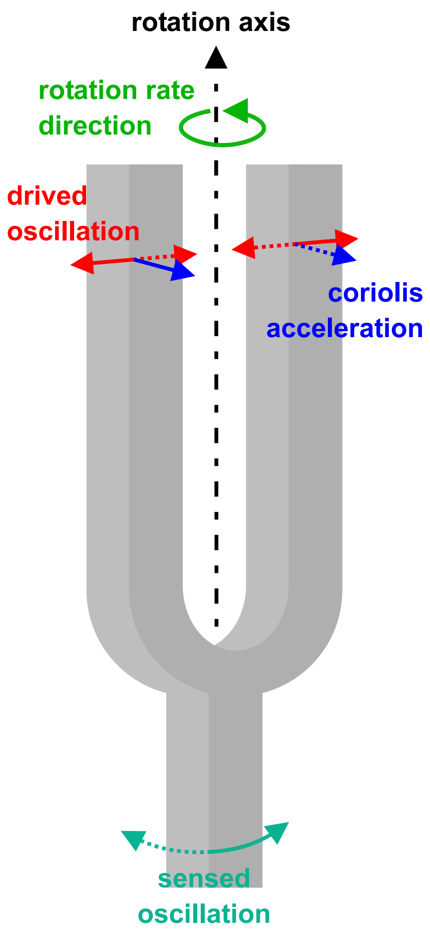


Figure 2.1: Tuning-fork Gyroscope

olis force [12]

$$\mathbf{F}_c = m\mathbf{a}_c = -2m\boldsymbol{\omega} \times \mathbf{v}' \tag{2.1}$$

cause Coriolis acceleration $\mathbf{a}_c$ of tines. Because Coriolis acceleration has opposite sign in each half-period, stem of fork is torsional oscillating (cyan arrows). This oscillation is proportional to angular rate.

The rest of MVG measure angle directly. This type is based on behaviour similar to Faucault pendulum. Faucault pendulum has two basic modes of motion. The modes are north-south and west-east swinging. The pendulum stays in north-south mode when it is not subject to rotation. The energy of pendulum will be transferred from the north-south mode to the west-east mode, if it is rotated. The transfer of energy between modes is proportional to angle of rotation. Angle measuring MVG can be based on isotropic resonator with 2 DoF or axisymmetric isotropic shell.

2 DoF isotropic resonator direct angle measurement is based on observation of vibration pattern itself. The model of mass-spring structure of 2 DoF is shown in figure 2.2. The input rotation angle θ_z is measured directly from precession of oscillation pattern.

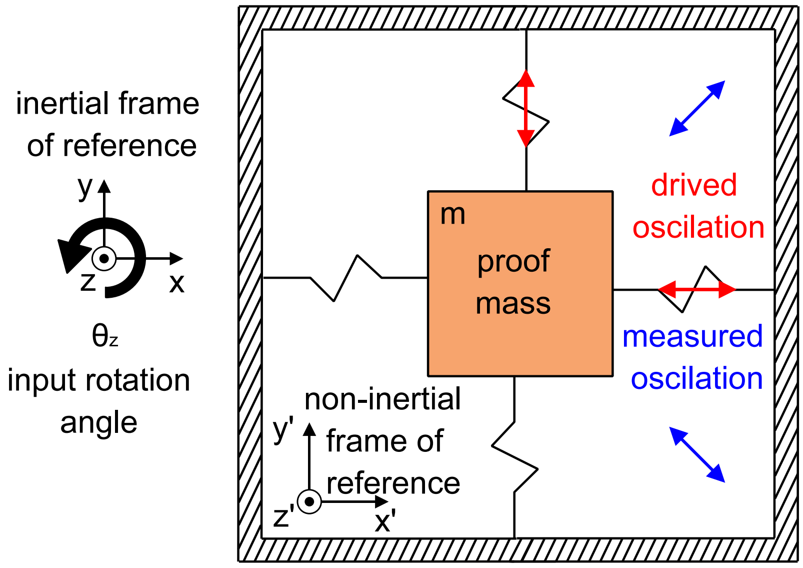


Figure 2.2: 2 DoF isotropic resonator model

Angle measuring MVG are more precise then angular rate measuring MVG. The precision can even reach inertial grade. However, they are also more expensive and thus not used in consumer grade IMU.

2.2.2 Accelerometers

Accelerometers measure inertial force, which is generated by accelerating proof mass. An accelerometer is made of at least three components - a proof mass, a suspension holding the mass and a pickoff, which relates an output signal to the induced acceleration [11]. This structure is shown in figure 2.3.

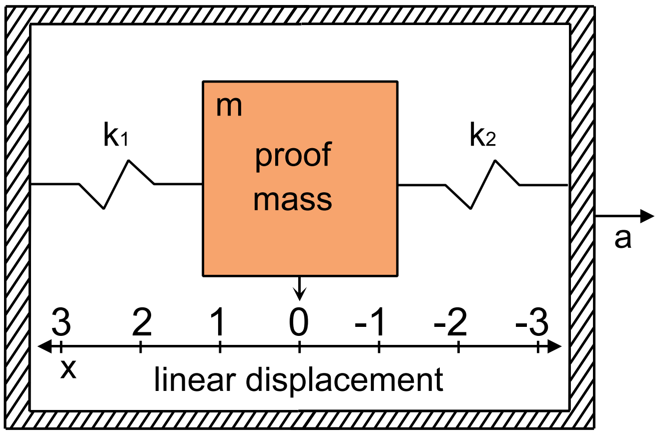


Figure 2.3: Basic accelerometer structure

The equilibrium of proof mass linear position is set to be zero displacement. With assumption, that suspension obeys Hooke's law and does not exceed its elastic limit, it can be written that

$$-ma = F = -kx = -(k_1 + k_2)x, \tag{2.2}$$

$$a = \frac{k}{m}x, \tag{2.3}$$

where m is a constant proof mass, a is an acceleration of sensor along sensitive axis, F is a reaction force applied on the proof mass, k, k_1, k_2 are constants of suspension and x is a linear displacement.

Accelerometers can be classified into several groups according to a transduction principle they use in order to convert displacement of proof mass to electrical signal. The first commercial accelerometers were resistance-bridge-type manufactured in 1920's using a carbon rings in compression-tension Wheatstone half-bridge [23]. The first micromachined and commercialized inertial sensors were based on piezoresistive principle [11]. In relation to compression/tension, a piezoresistor converts enlongation/shortening to a resistance amplification/reduction. The resistance change is then transferred to a voltage change using bridge techniques. Another used principles nowadays are piezoelectric, capacitive, optical and tunneling current sensing.

2.2.3 Magnetometers

Magnetometers also known as compasses can be based on several principles. This section do not aspire to explain them all, but to mention the most significant principles. The first realized way how to measure magnetic field was using the Hall effect. The Hall effect is named after Edwin Hall and was discovered in 1879 [20]. Hall sensors use Lorentz force

$$\mathbf{F}_L = q\,(\mathbf{E} + \mathbf{v} \times \mathbf{B})\,. \tag{2.4}$$

Lorentz force specifies the influence of electric and magnetic field on a electric charge. Because MEMS Hall sensors are made from a semiconductor, the electric charge means both negative electrons a positive holes.
A small drift current occurs due to Lorentz force. The drift current produces Hall voltage, which is related to surrounding magnetic field. The basic sensor scheme is shown on image 2.4.

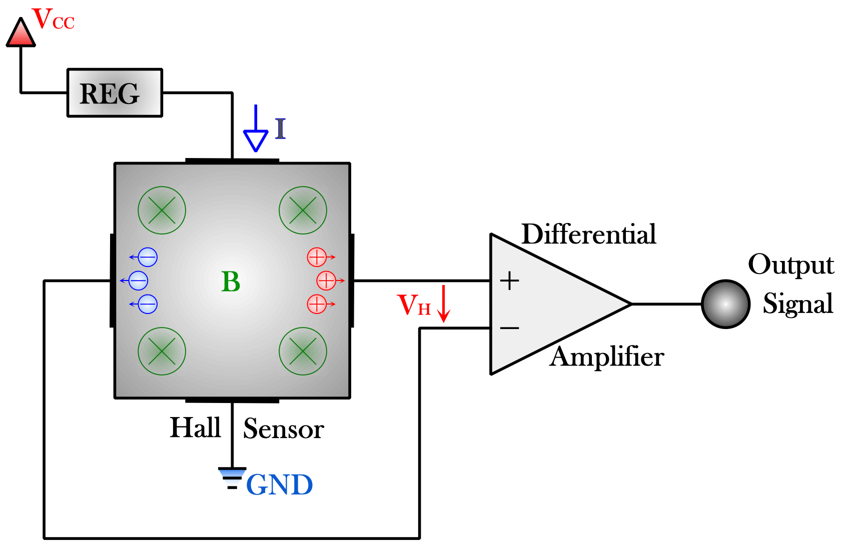


Figure 2.4: Hall sensor scheme

In 1980s, Giant Magnetoresistance (GMR) was discovered by Albert Fert and Peter Grünberg[2]. GMR is based on quantum phenomena of an electron spin. The most frequently used type of GMR is a spin-valve[3] [21]. Though GMR is more precise than Hall sensor, it's not used as a smartphone MEMS sensor because of higher price.

Anisotropic Magnetoresistance (AMR) sensor was invented in 1990, but the AMR effect of nickel and iron had been known since 1857 [2]. AMR effect is based on the fact that electric resistance is higher in direction of magnetisation than in direction perpendicular to direction of magnetisation. AMR sensors uses permalloy (alloy of Fe and Ni) on silicon substrate. Permalloy is magnetized in $\mathbf{x}$ direction in figure 2.5.

[2]A. Fert and Grünberg were awarded with the Nobel Prize in Physics 2007 `https://www.nobelprize.org/nobel_prizes/physics/laureates/2007/popular-physicsprize2007.pdf`

[3]Research under key word spintronics.

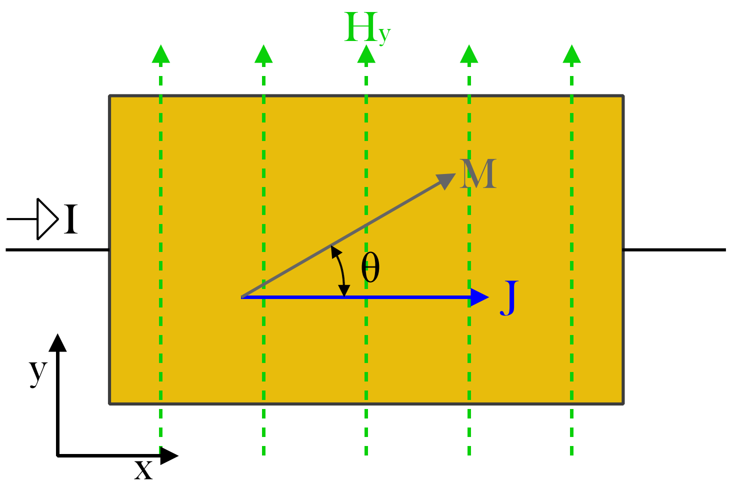


Figure 2.5: ARM sensor model

The magnetization **M** is rotated because of influence of magnetic field strength $\mathbf{H}_y$ in **y** direction. Magnetic field strength **H** is related to magnetic field **B**:

$$\mathbf{B} = \mu_m \mathbf{H}. \tag{2.5}$$

The anisotropic magnetoresistance is decreasing with θ approaching ± 90 degrees. Therefore, ARM sensor using single permalloy cannot determine exact direction and bridge techniques must be used. Usually ARM sensor is made as full-bridge with permalloy interleaved with aluminium according to Barber pole design.

Magnetoimpedance (MI) sensor [18] consists of an MI element, an current supply and signal processing circuitry. First part of MI element is a magnetosensitive wire formed of an amorphous soft magnetic alloy with zero magnetostriction. Magnetostriction is change in dimensions of a ferromagnetic materials subjected to a magnetic field. Second part is a detection coil around the wire separated by electric insulator. A detecting voltage generated in the detection coil is related to external magnetic field as long as the current supply provides an alternating current with frequency in hundreds of Megahertz.

Hall effect, AMR and MI are principles frequently used in smartphone MEMS IMU. But a lot of potential principles such as GMR, tunneling magnetoresistance (TMR) [21], giant magnetoimpedance (GMI) [2] , magnetoelectric (ME) effect [2] can be used in future smartphone compass sensors.

2.3 IMU Used in Smartphones

Almost every contemporary smartphone is equipped with accelerometer, magnetometer and gyroscope. These sensors are used in computer interfaces in various games and applications. Furthermore, these sensors are used to enhance GPS navigation performance.

The quality of a sensor is corresponding to its price. Because the fundamental purpose of smartphone is not inertial navigation, the IMU used is limited by a fraction of a targeted price. Targeted smartphone dimension and weight also create constrains on used sensors. The result is usage of consumer grade accelerometer and gyroscope. The example specification of consumer grade in comparison to tactical and inertial grade are shown in figure 2.1 [13]. The price, dimensions and weight are also limiting magnetometer, consequently we can find only compasses based on Hall effect, AMR and MI in smartphones.

Parameter		Unit	Inertial	Tactical	Consumer
Accelerometer	Pre-calibration bias	mg	0.050	1	10
	Scale factor error	%	0.002	0.03	1
	Misalignment	mrad	0.05	0.5	-
	Cross-axis sensitivity	%	-	-	1
Gyroscope	Pre-calibration bias	°/h	0.005	1	1000
	Bias instability	°/h	0.0035	1	25
	Angular random walk	$°/\sqrt{h}$	0.005	0.125	2
	Scale factor error	%	0.001	0.15	1
	Misalignment	mrad	0.01	0.5	-
	Cross-axis sensitivity	%	-	-	1

Table 2.1: Example specification for IMUs of different quality grades [13]

Because of limited quality IMU sensors, the manufacturers of sensors include features to improve overall performance such as calibration or in-line signal processing. Some sensors already have the ability to fuse data from an external sensor or multiple sensors.

In our experiment we used two different smartphones - LG Optimus 2X and Samsung Galaxy S5. They both have 9 DoF IMU. The components of Samsung Galaxy S5 IMU was possible to find out through Android OS. However, in case of LG Optimus 2X IMU disassembly of the phone was necessary in order to find out all used components.

The gyroscopes used in smartphones are Invensense MPU-6500 in case of Samsung Galaxy S5 and MPU-3050 in case of LG Optimus 2X. Their main features are compared in table 2.2.

Specification	Units	Invensense MPU-6500	Invensense MPU-3050
Full-scale Range	°/s	±250/500/ 1000/2000	±250/500/ 1000/2000
Gyroscope ADC word length	bits	16	16
Sensitivity scale Factor Tolerance (25 °C)	%	typical ±3	typical ±2
Sensitivity scale Factor Variation Over Temperature (−40 to +85 °C)	%	typical ±4	typical ±2
Nonlinearity	%	typical ±0.1	typical 0.2
Cross-Axis Sensitivity	%	typical ±2	typical 2
Initial Zero-Rate Output Tolerance	°/s	typical ±5	typical ±20
Zero-Rate Output Variation Over Temperature (−40 to +85 °C)	°/s/°C	typical ±0.24	typical ±0.03

Table 2.2: Gyroscopes comparison [10][9]

The used accelerometers are Invensense MPU-6500 in Samsung Galaxy S5 and Kionix KXTF9 edition accelerometer in LG Optimus 2X. The Kionix KXTF9 edition accelerometer is not specified so it can be KXTF9-1026, KXTF9-2050, KXTF9-4100. However, the only difference between their specifications is in Self Test Output change on Activation, which is not critical for any of our tests. The most important specifications of accelerometers are specified in table 2.3.

Specification	Units	Invensense MPU-6500	Kionix KXTF9
Full-scale Range	g	±2/4/8/16	±2/4/8/16
ADC word length	bits	16	12
Sensitivity scale Factor Variation Over Temperature (−40 to +85 °C)	%/°C	typ. ±0.026	typ. ±0.01(xy) ±0.03(z)
Nonlinearity	% of FS	-	typ. 1
Nonlinearity Best fit Straight Line	%	typ. ±0.5	-
Cross-Axis Sensitivity	%	typ. ±2	typ. 2
Initial Zero-g Offset	mg	-	max. ±125
Zero-g Variation Over Temperature (−40 to +85 °C)	mg/°C	typ. ±0.64(xy) ±1(z)	typ. ±0.7(xy) ±0.4(z)

Table 2.3: Accelerometers comparison [10][8]

The magnetometers incorporated in smartphones are AKM AK09911 in Samsung Galaxy S5 and AICHI STEEL AMI304 in LG Optimus 2X. AKM AKO9911 is based on high sensitive Hall sensor technology. AICHI STEEL AMI304 is Magnetoinduktance based compass. Available magnetic characteristics are shown in table 2.4.

Specification	Units	AKM AK09911	AICHI STEEL AMI304
Full-scale Range	μT	min. ±4900	±600
ADC word length	bits	14	12
Linearity of Range ±150μT	% of FS	-	typ. 0.5
Magnetic sensor sensitivity	$\mu T/LSB$	typ. 0.63	typ. 0.17
Initial Offset (25 °C)	LSB	max. ±500	-
Output Offset at Zero-μT (25 °C)	LSB	-	0
Output Offset temperature drift at Zero-μT (25 °C)	$\mu T/°C$	-	max. ±0.3

Table 2.4: Magnetometer comparison [7][6]

The important feature of both Samsung and LG IMUs is, that used Invensense Motion Processing Unit has sensor fusion. This feature allows us to skip design of our own gyroscope, accelerometer and magnetometer fusion. Thus in order to track head orientation we don't need to care about accelerometer and magnetometer any more. It should be sufficient to use gyroscope output after sensor fusion.

2.4 Experiment

Our experiment with a smartphone IMU was based on logging data in tested smartphone. Because smartphones have accelerometer, gyroscope and magnetometer fusion, we did not concerned ourselves with sensor fusion. We let used Invensense Motion Processing Unit (MPU-6500 and MPU-3050) take care of it. The smartphone was rotated and an input rotation measured. The input rotation in a measurement was three times 360° in positive angle direction (counter-clockwise) and in negative angle direction (clockwise). The measurements were done for three different angular speeds ω. The three measurements with different speeds were done for three axes parallel to `SensorEvent` API [3] axes x, y, and z shown picture 2.6.

Android application named *Naviin* was made to log sensor data in tested smartphones. This application is able to find out all sensors which are present in smartphone. Furthermore *Naviin* is able to capture selected sensors output data in selected rate. The selected sensors output data are then saved to comma-separated value (CSV) format file . The created *Naviin* application is more closely described in subsection 2.4.1.

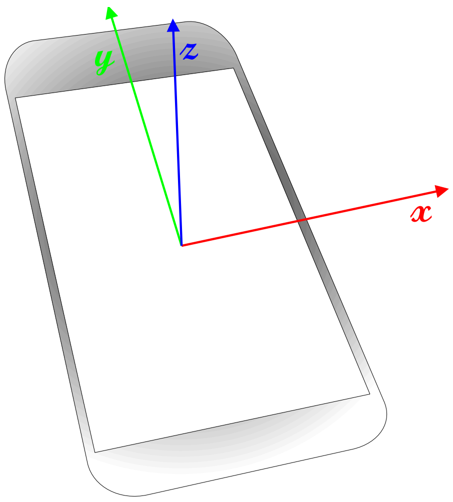


Figure 2.6: Definition of Coordinate System used by the SensorEvent API

The input rotation was generated by DC motor with an angular position control and measuring. The angular position control and measuring was done using PiKRON MARS 2 and Matlab® m-script *phoneOnMars* that we created. More about input rotation control and measuring in subsection 2.4.2.

The input rotation and output sensor data were compared. The comparison was done using output sensor data graphs generated by created Matlab® function *sensorDataPlotter* with graphic user interface and part of *phoneOnMars* script plotting input rotation. However, the input rotation and the output sensor data measuring were not synchronized so manual time-shift was necessary in order to get correct results.

2.4.1 *Naviin* - IMU Data Capture Application

Three-axis gyroscope, three-axis accelerometer and three-axis magnetometer data were collected through *Naviin*. The application provides these features:

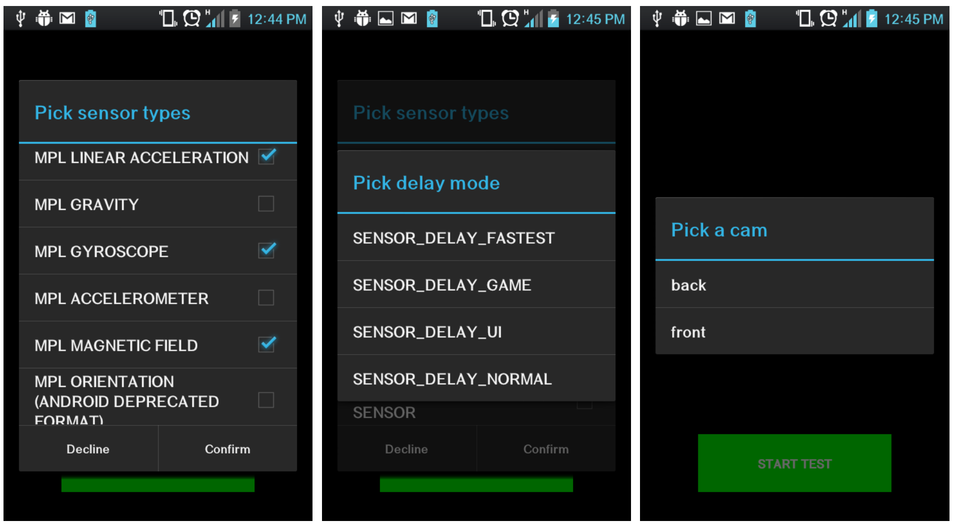
- Capture output data from any selected sensor in smartphone
- Save captured sensor data to CSV file
- Record video

The application starts with brief settings of provided features. The settings consists of three dialogs. The dialogs are shown on 2.7.

The first dialog requests to choose sensors whose data will be captured. The dialog is made as `DialogFragment` with radiobuttons 2.7*a*. The radiobuttons are labeled with names of sensors in smartphone. The application is able to recognize `Sensors` available in Android smartphone using `SensorManager` API.

The second dialog require to choose delay between individual data logs. The dialog *Pick delay mode* is made also `DialogFragment` 2.7*b*, but only one option is possible. The options are labeled with names of delay constants used in `SensorManager` API. The delay constants do not specifies any time period between individual logs. They are considered to be four duration levels, with given duration order shown in table 2.5. However, sometimes more delay options next to each other in Delay mode table 2.5 can result in same logging period.

The last necessary dialog asks which camera to use, in case of multiple cameras present. The dialog *Pick a cam* again made as `DialogFragment` 2.7*c* has only one possible option. The options are *back* and *front* related to back-facing and front-facing (selfie) cameras. The chosen camera is then used to video recording.



(a) : Sensor menu **(b) :** Delay menu **(c) :** Camera menu

Figure 2.7: Print screens of *Naviin* settings

Mode	Duration
SENSOR_DELAY_FASTEST	The shortest
SENSOR_DELAY_GAME	Shorter
SENSOR_DELAY_UI	Longer
SENSOR_DELAY_NORMAL	The longest

Table 2.5: Delay mode table

When the initial setting is finished, a process composed of a sensor data logging and video recording can be started in *Navin* `Activity`. The process is launched by pressing green button with *START TEST* label. The process is finished after pressing red button with *STOP TEST* label. One measurement consists of three stages.

In first stage, after pressing the green button, *Naviin* `Activity` starts a `Service` named *SensorLoggerService* by `Intent` with an extra. The extra consists of selected delay, selected sensors and absolute path to directory prepared for sensor logfile. Then the video file *videoOutput.mp4* is created and camera recording is started. After that the `Service` creates *Description.txt* text file with available sensor description from `Sensor` API. In the end of the first stage button turns red and changes label to *STOP TEST*.

In second stage, CSV file named *Logfile.txt* for sensor logs is created by the *SensorLoggerService*. The *SensorLoggerService* also registers himself as `SensorEventListener` for selected `Sensors` at `SensorManager`. At the rest of second stage the `SensorEventListener` starts `AsyncTask` which writes each `SensorEvent` to *Logfile.txt* CSV file of all logs.

The third stage takes place right after pressing a red button with *STOP TEST* label. The *logfile.txt* is closed. Then `SensorEventListener` is unregistered and `Service` is destroyed. After that camera recording is finished and the video file *videoOutput.mp4* is saved. In the end of the third stage button turns green and changes label to *START TEST*.

The output files from measurement n can be found in a folder *sample n*. The folders *sample n* are stored in *Naviin Measured data* on internal SD cart.

2.4.2 Rotation Control and Measurement

Rotation was done by electronic control unit PiKRON MARS 2 controlling DC motor position. MAXON RE 70W / 24 V DC motor was used in order to achieve high precision. The DC motor was combined with HP HEDL 5540 incremental position sensor using RS-422. MARS manufacturer claims [4] that with these components the accuracy is 1/2000 of one revolve. Therefore, with gears we are able to distinguish 1/10000 of one revolve of the rotation table 2.8

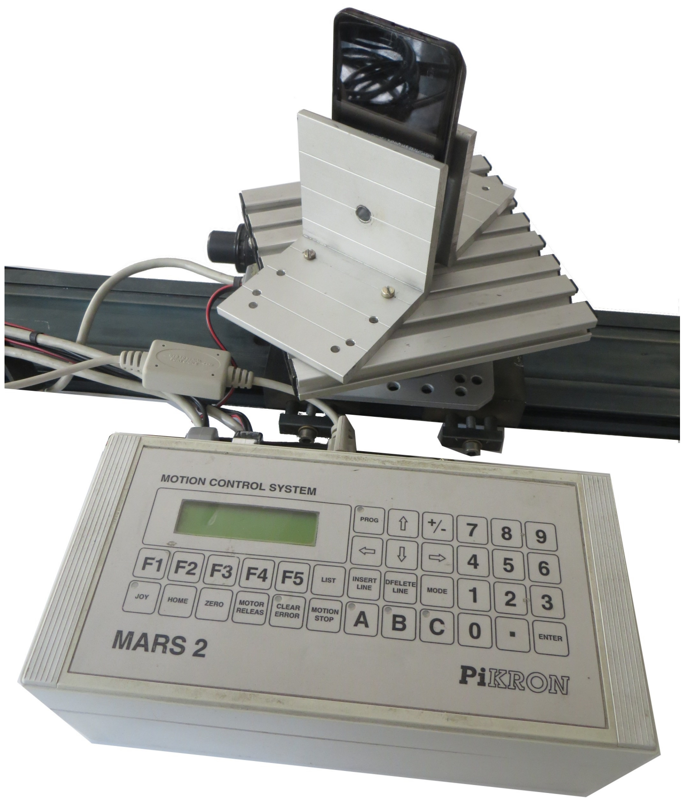


Figure 2.8: Measuring setup - MARS 2, rotation table with

Sending commands and receiving position was processed via RS-232. RS-232 was configured to 8 data-bits, 2 stop-bits, no parity and highest communication speed 19,200 Bd. Matlab® R2014b m-script *phoneOnMars* was used to send commands with description of rotation to execute. The m-script also received, recorded and plotted motor angular position data. However, MARS 2 does not uses angle in radians nor degrees, but instead uses "MARS units", where:

$$360 \text{ degrees} = 2\pi \text{ rad} = 10 \text{ "MARS units"} \tag{2.6}$$

The control unit was also used to set a angular rate. The angular rate was set using M_S parameter. Three measured speeds were set as

$$M_S \in \{100, 400, 1000\}, \tag{2.7}$$

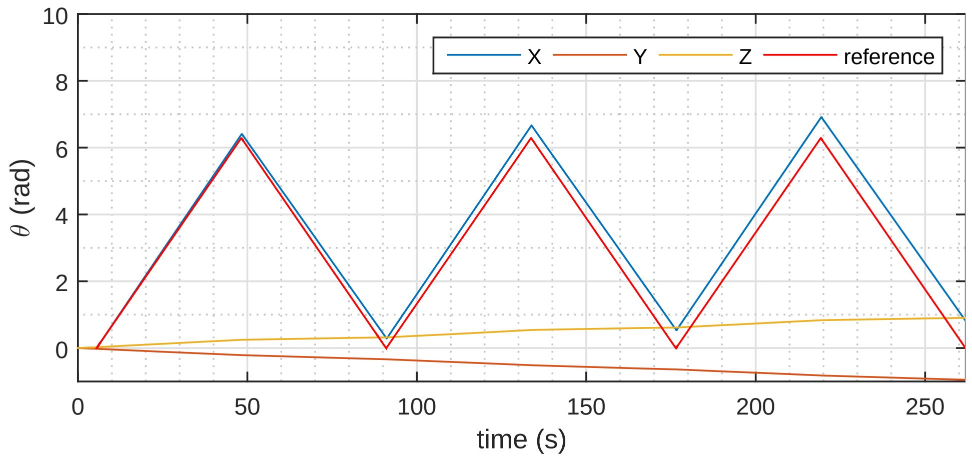
$$\omega \approx 0.00147 \cdot M_S \text{ rad} \cdot \mathrm{s}^-1, \tag{2.8}$$

$$\omega \in \{0.147, 0.588, 1.47\} \text{ rad} \cdot \mathrm{s}^-1. \tag{2.9}$$

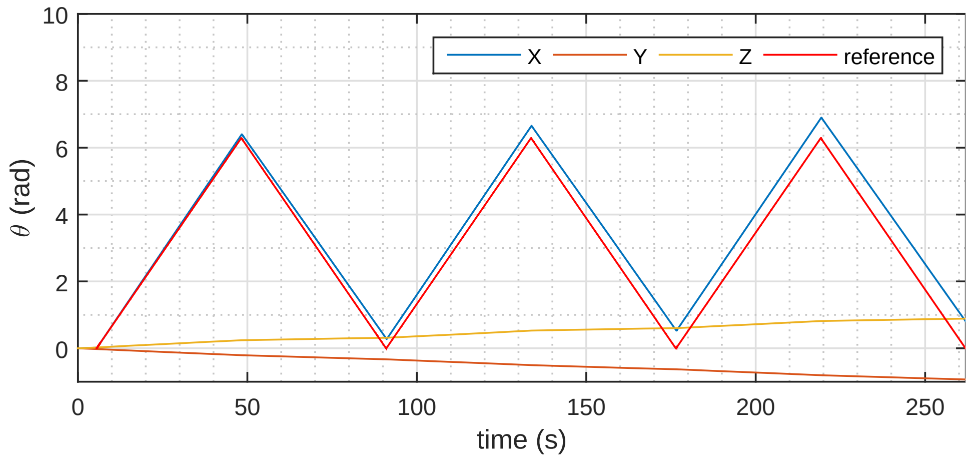
2.5 Analysis of Measured Data

This section describes results using exemplary measured samples. However, all measured samples from Samsung Galaxy S5 are in appendix *A* and all measured samples from LG Optimus 2X are in appendix *B*.

We can see from the logs that Samsung Galaxy S5 has two software gyroscope type sensors available. Those software sensors are both provide by Invensense MPU-6500. Their names are *MPU6500 Uncalibrated Gyroscope Sensor* and *MPU6500 Gyroscope Sensor*.



(a) : $M_S = 100$ - MPU6500 Gyroscope Sensor



(b) : $M_S = 100$ - MPU6500 Uncalibrated Gyroscope Sensor

Figure 2.9: Example from Samsung x-axis rotation test

The most surprising finding is, that both software sensors data results in the same graphs shown in figures 2.9*a* and 2.9*b*. Therefore, we can assume, that Motion Processing Unit MPU-6500 was not calibrated, because *MPU6500*

Uncalibrated Gyroscope Sensor data equals *MPU6500 Gyroscope Sensor* data.

Furthermore, the measured angle drifts in all samples shown in appendix *A*. The drifts are the most visible in graphs of slow speed samples $M_S = 100$, which has longest duration e.g. figures 2.9*a*, 2.9*b*. This characteristic is with high probability caused by absence of initial zero-rate output calibration.

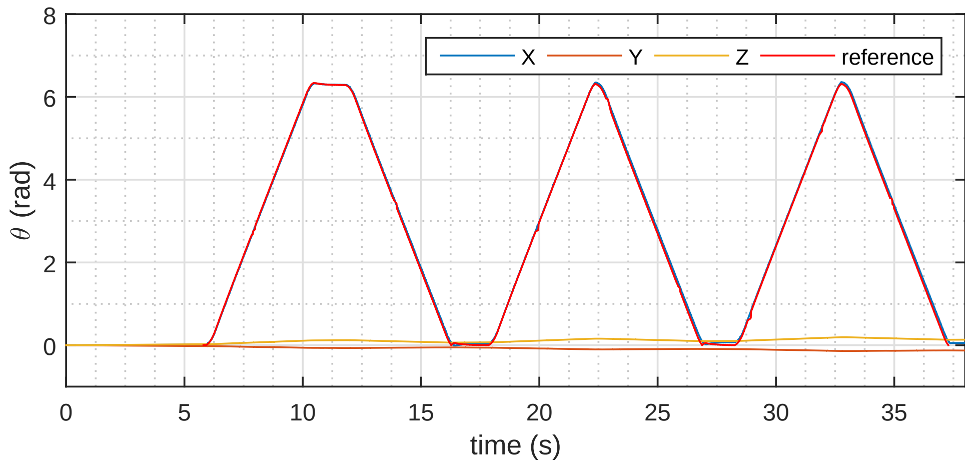
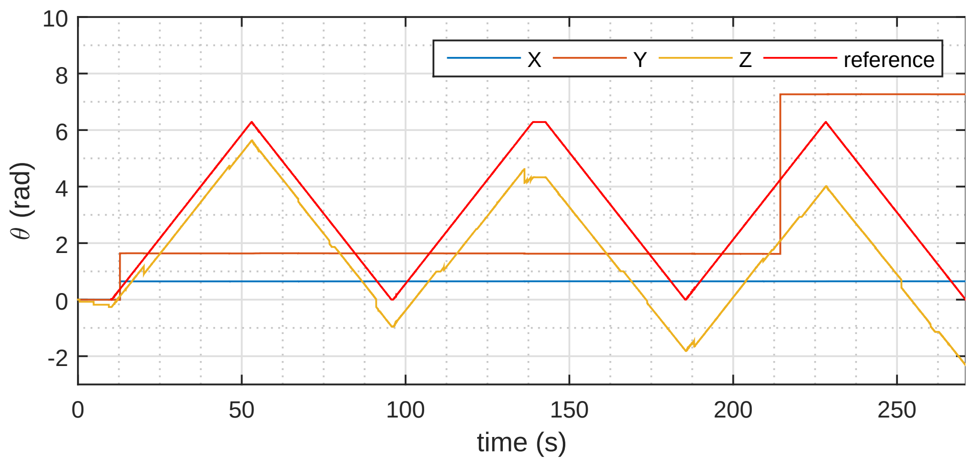


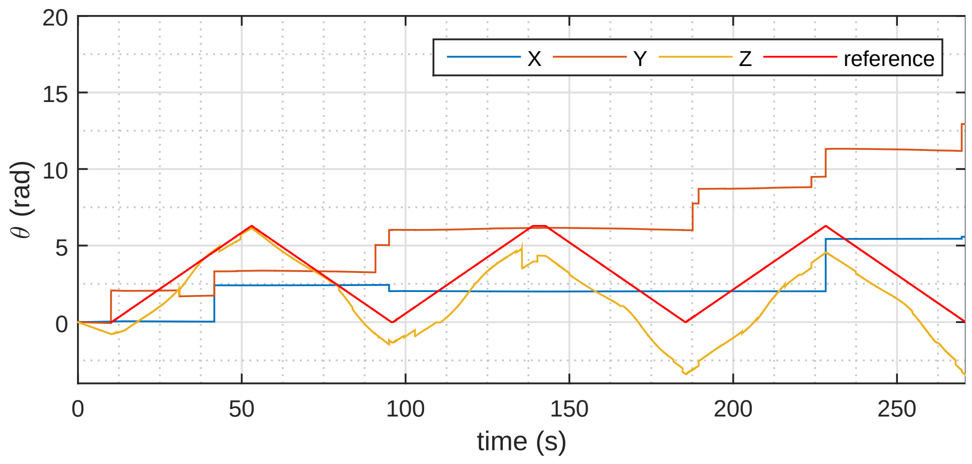
Figure 2.10: Example from Samsung x-axis rotation test: $M_S = 1000$ - MPU6500 Gyroscope Sensor

Moreover, the cross-axis sensitivity occurs not to be filtered out by sensor fusion. The cross-axis sensitivity is the most visible in graphs of fast rotation $M_S = 1000$ e.g. fig. 2.10. That is because a measured angular rate about an axis A is equal to sum of the angular rates about the rest of axes multiplied by a coefficient in range ± 0.02 and an actual angular rate about the axis A.

The logs from LG Optimus 2X tells us that the smartphone has also two software gyroscopes available. They are named *Corrected Gyroscope Sensor* and *MPL Gyroscope*. The *MPL Gyroscope* is provided by Invensense MPU-3050, but *Corrected Gyroscope Sensor* provided by Google Inc. The MPL in *MPL Gyroscope* name stands for Motion Processing Library, which provides sensor fusion.



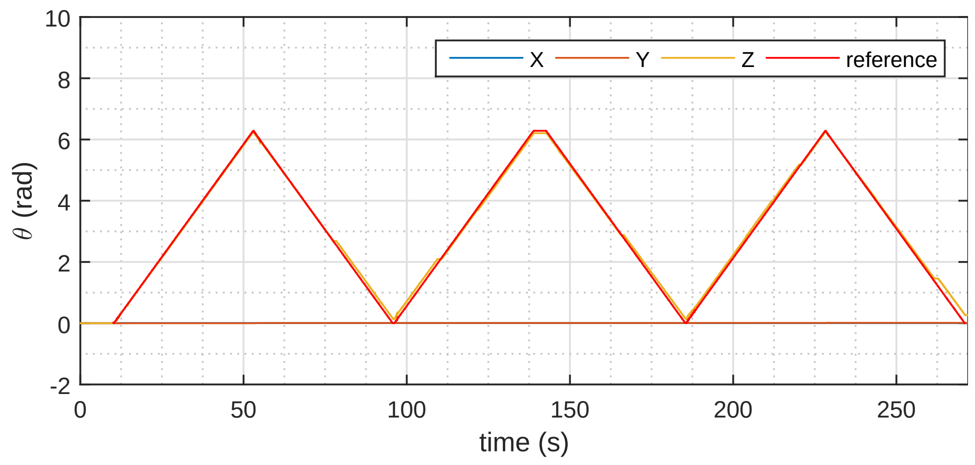
(a) : $M_S = 100$ - MPL Gyroscope



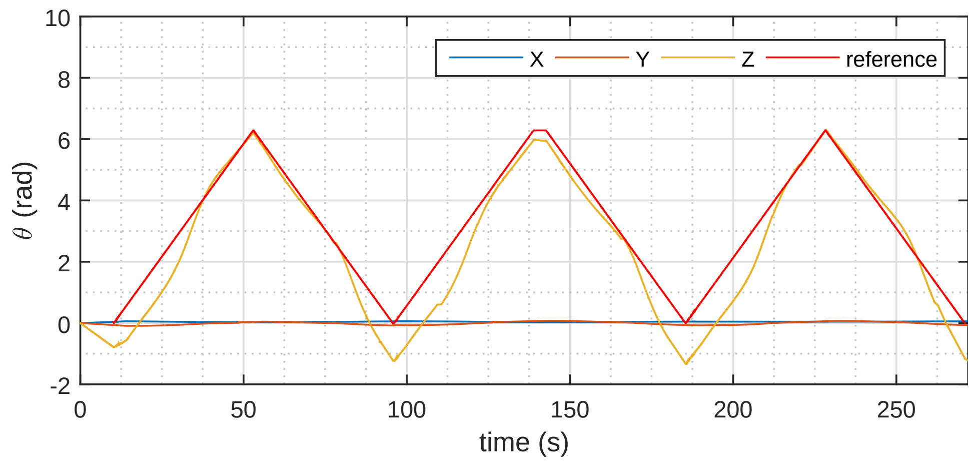
(b) : $M_S = 100$ - Corrected Gyroscope Sensor

Figure 2.11: Example from LG z-axis rotation test

The first thing we can notice in example graphs in figures 2.11a, 2.11b is that each graph X,Y and Z of both sensors contains discontinuities. Though fused *MPL Gyroscope* has less discontinuities than *Corrected Gyroscope Sensor*, it is still not able to filter them out on its own. Therefore we have analysed the output data from sensors and solved the problem with a simple solution. The solution is filter which will substitute triplet of X,Y and Z angular rates, if the absolute value of X,Y or Z angular rate will be bigger than 1.5 rad$\cdot$s^{-1}. If it is possible, the substitution uses average of previous and succeeding value. If there is no previous or succeeding value, the substitution is initial value (zero) or previous value respectively. Using this solution we get results shown in figures 2.12a and 2.12b.



(a) : $M_S = 100$ - MPL Gyroscope



(b) : $M_S = 100$ - Corrected Gyroscope Sensor

Figure 2.12: Example from LG z-axis rotation test with filter

We can see, that after adding filter of discontinuities, *MPL Gyroscope* follows input rotation very accurately. The *MPL Gyroscope* with filter do not drift. Moreover, the *MPL Gyroscope* is even able to suppress a cross-axis sensitivity e.g. figure 2.13, though it's specified to be 2%.

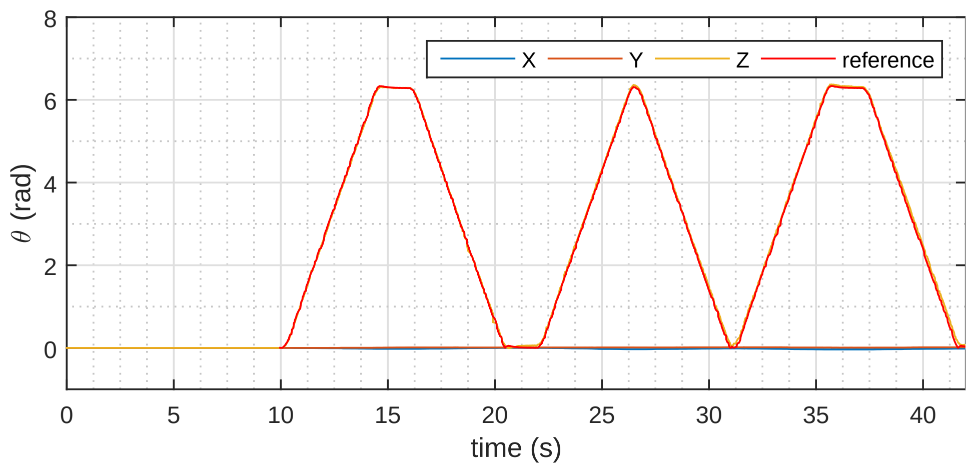


Figure 2.13: Example from LG z-axis rotation test with filter: $M_S = 1000$ - MPL Gyroscope

However, *Corrected Gyroscope Sensor* has problems to measure speed correctly, which results in the curves instead of lines which we can see in fig. 2.12*b*. But, this enables us to see difference between the fused *MPL Gyroscope* and the non-fused *Corrected Gyroscope Sensor*. The *Corrected Gyroscope Sensor* probably uses only initial zero-rate output calibration, because for example X and Y graph in fig 2.12*b* do not drift. Therefore the word *Corrected* in its name.

Chapter 3

Camera

A camera is an optical instrument which captures images or records video. If we want to augment the video, we will need to understand how camera works. This chapter is focused on describing a camera model and obtaining the model from camera calibration.

Firstly we will need to introduce a vocabulary prior to describing the camera model. Then we will describe the camera model and distortion model. Further, we will do a camera calibration in order to get camera models of Samsung Galaxy S5 and LG Optimus 2X back-facing cameras. In the end, we'll evaluate the results.

3.1 Introduction of vocabulary

This chapter uses terminology introduced in *Geometry of single view* lecture[17].

3.1.1 Terms and Definitions

- *Pinhole model* is the simplest (normalized) model of an perspective camera .
- *Projective space*
  - Consider $(d+1)$-dimensional vector space without its origin, $\mathbb{R}^{d+1} \setminus \{(0,0,...,0)\}$.
  - Define an equivalence relation

$$\begin{aligned} [x_1, x_2, ..., x_{d+1}]^T &\equiv \quad [x'_1, x'_2, ..., x'_{d+1}]^T , \\ \text{iff } \exists\alpha \neq 0 : [x_1, x_2, ..., x_{d+1}]^T &= \alpha\, [x'_1, x'_2, ..., x'_{d+1}]^T . \end{aligned}$$

  - *Projective space* $\mathbb{P}^d$ is quotient space of this equivalence relation.
- The projective plane $\mathbb{P}^2$
  - We will denote points in $\mathbb{P}^2$ by $\mathbf{u} = [u, v, w]^T$, lines (hyperplanes) in $\mathbb{P}^2$ by $\mathbf{l}$.

- Line through 2 points: $\mathbf{l} = \mathbf{x} \times \mathbf{y}$.
- Point as intersection of 2 lines: $\mathbf{x} = \mathbf{l} \times \mathbf{m}$.

- The projective plane $\mathbb{P}^3$
  - We will denote points in $\mathbb{P}^3$ by $\mathbf{X} = [X, Y, Z, W]^T$.
  - Planes are hyperplanes in $\mathbb{P}^3$.
  - A 3D line in $\mathbb{P}^3$ is an entity with no counterpart and has no elegant homogeneous representation by a vector like points and planes in $\mathbb{P}^3$.

- *Homogeneous coordinates* (also *Projective coordinates*)
  - Points in *projective space* are expressed in *homogeneous coordinates* $\mathbf{x} = \alpha\ [x'_1, x'_2, ..., x'_d, 1]^T$.

- *Homography* (*Collineation, Projective transformation*)
  - *Homography* is any mapping $\mathbb{P}^d \rightarrow \mathbb{P}^d$ linear in the embedding space $\mathbb{R}^{d+1}$.
  - *Homography* is defined up to unknown scale as $\mathbf{u}' \simeq \underline{\mathbf{H}}\mathbf{u}$, where $\underline{\mathbf{H}}$ is $(d+1) \times (d+1)$ matrix.
  - The *projective transformation* maps any triplet of collinear points to a triplet of collinear point,therefore the name *collineation*.
  - If $\underline{\mathbf{H}}$ is regular then distinct points are mapped to distinct points.
  - In $\mathbb{P}^2$, *collineation* is the most general transformation mapping lines to lines.

3.1.2 Basic relations and properties

- Relation between Euclidean and projective spaces
  - Consider Euclidean space $\mathbb{R}^d$.
  - Non-homogeneous coordinates represent a point in $\mathbb{R}^d$ occupying the plane in $\mathbb{R}^{d+1}$ with equation $x_{d+1} = 1$.
  - The one-to-one mapping from $\mathbb{R}^d$ to $\mathbb{P}^d$ is

$$[x_1, x_2, ..., x_d]^T \rightarrow [x_1, x_2, ..., x_d, 1]^T .$$

  - Points $[x_1, x_2, ..., x_d, 0]^T$ from projective space do not have Euclidean counterpart and represent points at infinity in a particular direction.
  - Consider $[x_1, x_2, ..., x_d, 0]^T$ as a limiting case of $[x_1, x_2, ..., x_d, \alpha]^T$ that is projectively equivalent to $[\frac{x_1}{\alpha}, \frac{x_2}{\alpha}, ..., \frac{x_d}{\alpha}, 1]^T$, and assume that $\alpha \rightarrow 0$.
  - This corresponds to a point in $\mathbb{R}^d$ going to infinity in the direction of the radius vector $[\frac{x_1}{\alpha}, \frac{x_2}{\alpha}, ..., \frac{x_d}{\alpha}]^T \in \mathbb{R}^d$.

- Homography vs. non-homography

  - Let us illustrate how the non-homogeneous 2D point $[u, v]^T$ (e.g. point in an image) is actually mapped to the non-homogeneous image point $[u', v']^T$ by $\underline{\mathbf{H}}$ using $\mathbf{u}' \simeq \underline{\mathbf{H}}\mathbf{u}$.

  - The equation $\mathbf{u}' \simeq \underline{\mathbf{H}}\mathbf{u}$ can be rewritten as

$$\alpha \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

  - Writing 1 in the third coordinate of $\mathbf{u}'$, we assume that $\mathbf{u}'$ is not a point at infinity, that is, $\alpha \neq 0$. To compute $[u', v']^T$, we need to eliminate the scale α. That yields the expressions

$$u' = \frac{h_{11}u + h_{12}v + h_{13}}{h_{31}u + h_{32}v + h_{33}}, \qquad v' = \frac{h_{21}u + h_{22}v + h_{23}}{h_{31}u + h_{32}v + h_{33}},$$

    which are necessary when homogeneous coordinates are not used.

  - The advantages of using homogeneous coordinates are that $\mathbf{u}' \simeq \underline{\mathbf{H}}\mathbf{u}$ is simpler, linear and can handle point $\mathbf{u}'$ at infinity.

3.2 Camera Model Description

This section uses two types of projective spaces $\mathbb{P}^2$, $\mathbb{P}^3$ and two Euclidean spaces $\mathbb{R}^2$, $\mathbb{R}^3$. The notation of an image point $\mathbf{u}$ is $\mathbf{u} = [u, v]^T$ in $\mathbb{R}^2$ or $\mathbf{u} = [u, v, w]^T$ in $\mathbb{P}^2$. The notation of a 3D scene point $\mathbf{X}$ is $\mathbf{X} = [X, Y, Z]^T$ in $\mathbb{R}^3$ or $\mathbf{X} = [X, Y, Z, W]^T$ in $\mathbb{P}^3$. The subscripts are used to distinguish different coordinate systems.

3.2.1 Perspective Camera Model

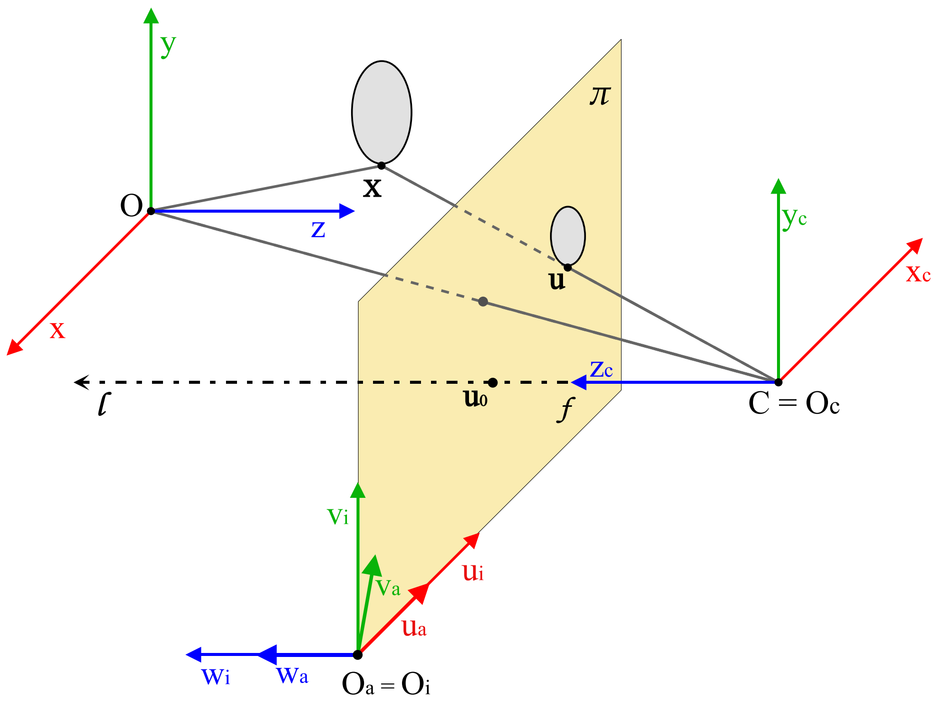


Figure 3.1: Perspective camera model description

Description of perspective camera model shown in figure 3.1 uses following coordinate systems:

- World Euclidean coordinate system with origin $\mathbf{O}$ and axes x, y and z.
- Camera Euclidean coordinate system with origin $\mathbf{O}_c$ and axes x_c, y_c and z_c.
- Image Euclidean coordinate system with origin $\mathbf{O}_i$ and axes x_i, y_i and z_i.
- Image affine coordinate system with origin $\mathbf{O}_a$ and axes x_a, y_a and z_a.

Transformation from World coordinate system to camera coordinate system is given by 3×3 rotation matrix $\underline{\mathbf{R}}$ and position vector $\mathbf{t}$ of camera coordinate system origin $\mathbf{C}$ in world coordinate system. Parameters $\underline{\mathbf{R}}$ and $\mathbf{t}$ are called extrinsic camera calibration parameters. The point $\mathbf{C}$ is focal point of a camera, lying on optical axis l. f is focal length of a camera, which can be used to determine the principal point $\mathbf{u}_0$ in the camera coordinate system $\mathbf{u}_{0c} = [0, 0, f]^T$. The principal point can be also determined as $\mathbf{u}_{0a} = [u_0, v_0]^T$ in image affine coordinate system.

Camera model can be described by a nonlinear transformation from $\mathbb{R}^3$ to $\mathbb{R}^2$, which can be written as

$$\mathbf{X}_c = \underline{\mathbf{R}}\left(\mathbf{X} - \mathbf{t}\right), \tag{3.1}$$

$$u = f\frac{U_c}{Z_c}, \tag{3.2}$$

$$v = f\frac{V_c}{Z_c}. \tag{3.3}$$

The non-homogeneous projection transformation is linear in focal length f, point coordinates X and Y but nonlinear in Z coordinate of the point.

However, a better way is to describe camera model as linear transformation from $\mathbb{P}^3$ to $\mathbb{P}^2$. The transformation can be factorized into three simple transformation between above defined coordinate systems. The first transformation is from world coordinates system $\mathbb{P}^3$ to camera coordinate system

$$\mathbf{X}_C = \begin{bmatrix} \underline{\mathbf{R}} & -\underline{\mathbf{R}}\mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{X}. \tag{3.4}$$

The second transformation is from camera coordinate system $\mathbb{P}^3$ to image Euclidean coordinate system $\mathbb{P}^2$, which corresponds with projection of normalized image plane ($f = 1$).

$$\mathbf{u}_i \simeq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{X}_c. \tag{3.5}$$

The last transformation is from image Euclidean coordinate system $\mathbb{P}^2$ to image affine coordinate system $\mathbb{P}^2$, also called affine mapping in the image plane

$$\mathbf{u} \simeq \underline{\mathbf{K}}\mathbf{u}_i = \begin{bmatrix} f & s & -u_0 \\ 0 & g & -v_0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{u}_i. \tag{3.6}$$

The upper triangular matrix $\underline{\mathbf{K}}$ is called intrinsic calibration matrix. The individual elements in $\underline{\mathbf{K}}$ are

- f - focal length, gives the scaling along the u axis.
- g - scaling along the v axis (often equal to focal length f).
- s - skew, $s = g\cos(\alpha)$, where α is angle from the x-axis to y-axis of the pixel elements in the CCD array . Parameter s is usually equal to 0, otherwise it would mean that the x- and y-axes are not perpendicular.
- $[u_0, v_0]$ are coordinates of the principal point.

The perspective camera projection in homogeneous coordinates is a product of the three factors above

$$\mathbf{u} \simeq \underline{\mathbf{K}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \underline{\mathbf{R}} & -\underline{\mathbf{R}}\mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{X} = \underline{\mathbf{K}} \left[\underline{\mathbf{R}} | - \underline{\mathbf{R}}\mathbf{t} \right] \mathbf{X} = \underline{\mathbf{M}}\mathbf{X}. \quad (3.7)$$

The perspective camera model is described by equation 3.7.

3.2.2 Real Camera Model

A real camera model differs from a perspective camera model in involvement of distortions. The real camera model improves the perspective camera model with incorporation of the lens distortions. We involved two most common distortions [14] - radial and tangential.

The real camera model uses the resulting image point $\mathbf{u} = [u, v, w]^T$ from perspective camera model. However, our real camera model uses only Euclidean space $\mathbb{R}^2$ of image plane. Therefore, transformation from $\mathbb{P}^2$ to $\mathbb{R}^2$ is necessary

$$u = \frac{u}{w}, \quad (3.8)$$

$$v = \frac{v}{w}. \quad (3.9)$$

Thus, every following u and v are coordinates of image point $\mathbf{u} = [u, v]^T$ in $\mathbb{R}^2$.

The radial distortion is caused by a simple lens manufacturing. The simple lens with radial distortion bends light rays farther from the center too much or too little. The case when the rays distant from the center are bent too much is also known as barrel distortion. Accordingly the case when the rays far from the center are bent too little is known as pincushion distortion. The position of pixel after radial distortion can be solved as

$$u_{rd} = u \left(1 + k_1 r^2 + k_2 r^4\right), \quad (3.10)$$

$$v_{rd} = v \left(1 + k_1 r^2 + k_2 r^4\right), \quad (3.11)$$

where r is a pixel radius $r = sqrtu^2 + v^2$, k_1 and k_2 are radial distortion coefficients.

The tangential distortion is caused by misalignment of the lens and an imaging plane, when the lens are not exactly parallel to the imaging plane. The position of pixel after tangential distortion can be calculated as

$$u_{td} = u + \delta_{tu} = u + \left[2p_1 v + p_2 \left(r^2 + 2u^2\right)\right], \quad (3.12)$$

$$v_{td} = u + \delta_{tv} = v + \left[p_1 \left(r^2 + 2v^2\right) + 2p_2 u\right], \quad (3.13)$$

where p_1 and p_2 are tangential distortion coefficients.

The real camera model can be described by equations

$$\begin{aligned}
\mathbf{X}_c &= [\underline{\mathbf{R}}| - \underline{\mathbf{R}}\mathbf{t}]\,\mathbf{X}, & (3.14)\\
\mathbf{u}_i &= \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix}, & (3.15)\\
\mathbf{u}_d &= \mathbf{u}_i\left(1 + k_1 r^2 + k_2 r^4\right) + \begin{bmatrix} 2p_1 v_i + p_2\left(r^2 + 2u_i^2\right) \\ p_1\left(r^2 + 2v_i^2\right) + 2p_2 u_i \end{bmatrix}, & (3.16)\\
\begin{bmatrix} \mathbf{u}_{rc} \\ 1 \end{bmatrix} &\simeq \underline{\mathbf{K}} \begin{bmatrix} \mathbf{u}_d \\ 1 \end{bmatrix}, & (3.17)
\end{aligned}$$

where $\mathbf{X}_c$ is position of 3D scene point with respect to camera coordinate system, $\mathbf{u}_i$ is point in the image Euclidean space from normalized projection, $\mathbf{u}_d$ is a normalized (Euclidean) image point after distortions and $\mathbf{u}_{rc}$ is resulting real camera image point.

3.3 Camera Calibration

Camera calibration is a process of intrinsic camera matrix estimation. We used open source Camera Calibration Toolbox for Matlab®[5] created by Jean-Yves Bouguet. This toolbox uses plane-based internal parameter estimation[19].

The plane-based internal parameter estimation is a method which doesn't require knowledge of calibration points position in space, but uses a planar calibration pattern with an assumption that the plane lies at world coordinates $Z = 0$. This assumption and RQ decomposition are sufficient to calibrate a camera. Thus, it is sufficient to use several pictures or video frames of a planar surface in different orientations. The planar surface requested by the Camera Calibration Toolbox is a calibration pattern composed of a checkerboard pattern. We used our Android application *Naviin* to record a video of a calibration pattern. Then we selected several frames from the video to use in calibration process. The frame example is shown in fig. 3.2.

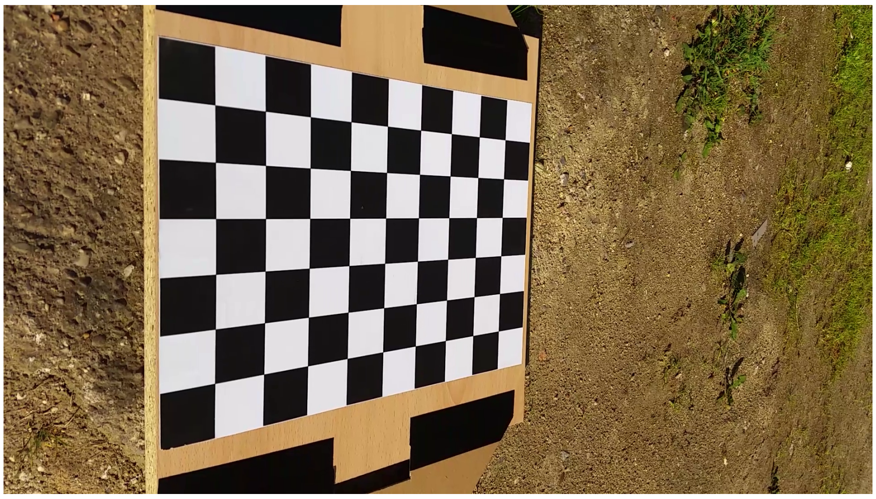


Figure 3.2: Used calibration pattern

The algorithm used by Camera Calibration Toolbox for Matlab® is inspired by algorithm proposed by Zhang[24] and uses internal camera model inspired from the model used by Heikkilä and Silvén[16].

3.4 Experimental Results

The camera calibration process gave us the intrinsic camera parameters and the distortion coefficients. The intrinsic camera parameters and distortion coefficients for LG Optimus 2X and Samsung Galaxy S5 are in tables 3.1*a* and 3.1*b* respectively. The uncertainties in uncertainty columns are approximately three times the standard deviations. We can see, that the uncertainties of the tangential distortion parameters are at least a half of estimated value and therefore they are highly unreliable. Thus we will not use them in our camera model.

Parameter	estimated value	uncertainty
f	1264	2
g	1261	2
$\cos(\alpha)$	-0.005	0.0004
u_0	656	4
v_0	372	3
k_1	0.047	0.008
k_2	-0.27	0.03
p_1	0.0014	0.0007
p_2	0.0002	0.0009

(a) : LG Optimus 2X

Parameter	estimated value	uncertainty
f	1563	2
g	1557	3
$\cos(\alpha)$	-0.005	0.0004
u_0	952	3
v_0	549	4
k_1	0.15	0.006
k_2	-0.33	0.01
p_1	0.0002	0.0008
p_2	0.0006	0.0008

(b) : Samsung Galaxy S5

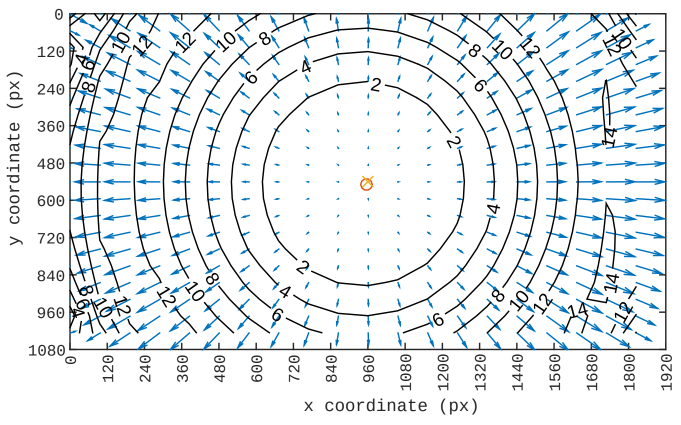
Table 3.1: Back-facing camera calibrated intrinsic parameters

Obtained intrinsic parameters can be rewritten into intrinsic calibration matrices K. The resulting matrices for LG Optimus 2X and Samsung Galaxy

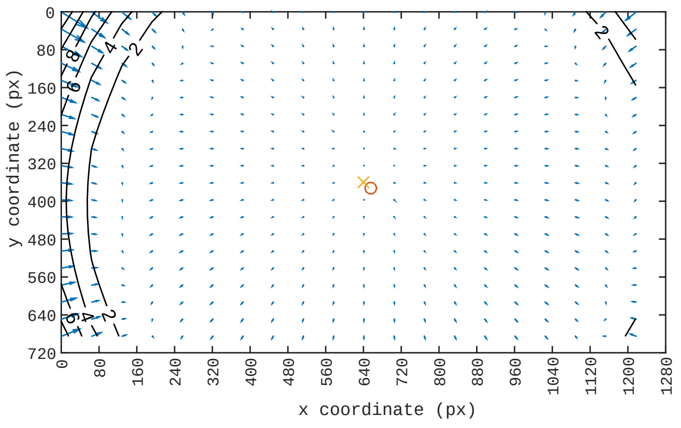
S5 are

$$\underline{\mathbf{K}}_{LG} = \begin{bmatrix} 1264 & -6 & 656 \\ 0 & 1261 & 372 \\ 0 & 0 & 1 \end{bmatrix}, \underline{\mathbf{K}}_{Samsung} = \begin{bmatrix} 1563 & -3 & 953 \\ 0 & 1558 & 550 \\ 0 & 0 & 1 \end{bmatrix}.$$

A graph of complete distortion model can be assembled from obtained distortion parameters. The distortion graphs for Samsung Galaxy S5 an LG Optimus 2X are shown in fig. 3.3.



(a) : Samsung Galaxy S5



(b) : LG Optimus 2X

Figure 3.3: Complete distortion model of back-facing camera

The partial graphs of radial and tangential component of distortion can be found in appendices *D* and *E*.

Chapter 4

Augmented Video

At the last part of our work we use the *Naviin* application, that we made, to record a video and capture an output sensor data. Then we utilize the captured output sensor data and camera calibration for creating a video overlay appearing to be fixed in 3D scene. In the end, we evaluate the augmented video which we created.

4.1 Our *gyroToVideo* function

gyroToVideo is a Matlab® function we made to use a gyroscope data for a video augmentation. The *gyroToVideo* inputs are following files:

- *Logging file* - the file of CSV logs created by our *Naviin* Android application.
- *Calibration file* - the Matlab® mat-file containing results from a smartphone camera calibration made in Camera Calibration Toolbox for Matlab®.
- *Record start file* - the file created by our *Naviin* Android application which contains timestamps from just before and just after the *video* recording started.
- *Video* - the video which was recorded by our *Naviin* Android application with the smartphone camera, which we calibrated.

The *logging file*, *record start file* and *video* must be from the same measurement e.g. all come from *sample x* folder from x-th measurement with our *Naviin* Android application.

After opening all four files, *gyroToVideo* finds out available gyroscope logs in *logging file*. Then a `VideoReader` of the *video*, which shall be augmented, is created. Also a `VideoWriter` for writing an augmented output video. After loading of necessary inputs an augmentation process follows.

The augmentation process begins with an initialization of the most important five vectors and one matrix

- `t` - vector made of times in seconds corresponding to individual logs. The times in the vector can be measured either from just before/after *video* recording started or from first log of gyroscope.
- $\underline{\mathbf{R}}$ - 3×3 rotation matrix from initial camera coordinate system to camera coordinate system in time of a given frame capturing. The $\underline{\mathbf{R}}$ is initialized to identity matrix $\underline{\mathbf{I}}$.
- `qX` - vector made of the first coordinates of the quaternions describing spatial orientation changes between individual logs.
- `qY` - vector made of the second coordinates of the quaternions describing spatial orientation changes between individual logs.
- `qZ` - vector made of the third coordinates of the quaternions describing spatial orientation changes between individual logs.
- `qW` - vector made of the forth coordinates of the quaternions describing spatial orientation changes between individual logs.

The vectors of the coordinates can be initialized coordinate after coordinate in loop, or at once by using vectorization. The coordinates `qX`, `qY`, `qZ` and `qW` of individual quaternions are calculated using following formulas

$$\mathtt{qX} = \frac{\bar{\omega}_x \sin\left(\frac{\Delta_t \|\bar{\boldsymbol{\omega}}\|}{2}\right)}{\|\bar{\boldsymbol{\omega}}\|}, \tag{4.1}$$

$$\mathtt{qY} = \frac{\bar{\omega}_x \sin\left(\frac{\Delta_t \|\bar{\boldsymbol{\omega}}\|}{2}\right)}{\|\bar{\boldsymbol{\omega}}\|}, \tag{4.2}$$

$$\mathtt{qZ} = \frac{\bar{\omega}_x \sin\left(\frac{\Delta_t \|\bar{\boldsymbol{\omega}}\|}{2}\right)}{\|\bar{\boldsymbol{\omega}}\|}, \tag{4.3}$$

$$\mathtt{qW} = \cos\left(\frac{\Delta_t \|\bar{\boldsymbol{\omega}}\|}{2}\right), \tag{4.4}$$

where used variables are:

- $\bar{\omega}_x$ - the average rotation about x axis between n-th and $(n+1)$-th logs in radians.
- $\bar{\omega}_y$ - the average rotation about y axis between n-th and $(n+1)$-th logs in radians.
- $\bar{\omega}_z$ - the average rotation about z axis between n-th and $(n+1)$-th logs in radians.
- Δ_t - the time between n-th and $(n+1)$-th logs in seconds.
- $\|\bar{\boldsymbol{\omega}}\|$ - the magnitude of average rotation vector

$$\bar{\boldsymbol{\omega}} = (\bar{\omega}_x, \bar{\omega}_y, \bar{\omega}_z)$$

between n-th and $(n+1)$-th logs in radians, calculated as

$$\|\bar{\boldsymbol{\omega}}\| = \sqrt{\bar{\omega}_x^2 + \bar{\omega}_y^2 + \bar{\omega}_z^2}.$$

After the initialization, a main part of augmentation process comes. The main part is a loop which can be divided into three functional segments - loading a frame with updating $\underline{\mathbf{R}}$, computing a pixel corresponding to direction of initial camera facing and drawing a marker into the frame. The loop ends when all frames are augmented.

4.1.1 Loading Frames and Updating of the Rotation Matrix

The first part of the loop starts with loading a frame from the `VideoWriter`. This event updates fields of `VideoWriter` such as *timestamp*, which we need to update the rotation matrix.

When we know the *timestamp* of a new frame, we can start an updating of $\underline{\mathbf{R}}$. The updating process uses a variable `k` as pointer to element of `t`, `qX`, `qY`, `qZ` and `qW` which should be used. Thus, the `k`-th elements of `qX`, `qY`, `qZ` and `qW` are used to update $\underline{\mathbf{R}}$ while the `k` is smaller than number of available logs, `k`-th element of `t` is greater than zero and smaller than *timestamp*. The update of $\underline{\mathbf{R}}$ is done by using conversion from quaternion to rotation matrix $\underline{\mathbf{R}}_q$, and following matrix multiplication:

$$\underline{\mathbf{R}}_q = \begin{bmatrix} 1 - 2\mathtt{qY}_k^2 + 2\mathtt{qZ}_k^2 & 2\mathtt{qX}_k\mathtt{qY}_k + 2\mathtt{qW}_k\mathtt{qZ}_k & 2\mathtt{qX}_k\mathtt{qZ}_k + 2\mathtt{qW}_k\mathtt{qY}_k \\ 2\mathtt{qX}_k\mathtt{qY}_k + 2\mathtt{qW}_k\mathtt{qZ}_k & 1 - 2\mathtt{qX}_k^2 + 2\mathtt{qZ}_k^2 & 2\mathtt{qY}_k\mathtt{qZ}_k + 2\mathtt{qW}_k\mathtt{qX}_k \\ 2\mathtt{qX}_k\mathtt{qZ}_k + 2\mathtt{qW}_k\mathtt{qY}_k & 2\mathtt{qY}_k\mathtt{qZ}_k + 2\mathtt{qW}_k\mathtt{qX}_k & 1 - 2\mathtt{qX}_k^2 + 2\mathtt{qY}_k^2 \end{bmatrix}, \tag{4.5}$$

$$\underline{\mathbf{R}} = \underline{\mathbf{R}}\underline{\mathbf{R}}_q, \tag{4.6}$$

where $\mathtt{qX}_k$, $\mathtt{qY}_k$, $\mathtt{qZ}_k$ and $\mathtt{qW}_k$ are `k`-th elements of vectors `qX`, `qY`, `qZ` and `qW`.

4.1.2 Computing Pixel Coordinates

When we know the rotation matrix between initial and actual camera coordinate system, we can use it to compute coordinates of a pixel corresponding to direction, where was camera initially facing. This direction can be written in homogeneous coordinates with respect to initial camera coordinate system as $\mathbf{X} = [0, 0, 1, 0]^T$. To obtain pixel position we use the described real camera model in subsection 3.2.2 composed from several equations. The first of them describes transformation of 3D scene in world coordinate system to camera coordinate system

$$\mathbf{X}_c = [\underline{\mathbf{R}}| - \underline{\mathbf{R}}\mathbf{t}]\,\mathbf{X},$$

where $\underline{\mathbf{R}}$ is rotation matrix from initial to actual camera coordinate system and $\mathbf{t}$ is translation between coordinate systems, which can be neglected, because the forth coordinate of $\mathbf{X}$ is a zero. The second equation is normalized perspective camera equation

$$\mathbf{u}_i = \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix}.$$

The third equation adds distortions into real camera model

$$\mathbf{u}_d = \mathbf{u}_i \left(1 + k_1 r^2 + k_2 r^4\right) + \begin{bmatrix} 2p_1 v_i + p_2 \left(r^2 + 2u_i^2\right) \\ p_1 \left(r^2 + 2v_i^2\right) + 2p_2 u_i \end{bmatrix},$$

where k_1 and k_2 are radial distortion coefficients of given camera and p_1, p_2 are tangential distortion coefficients. Coefficients k_1, k_2 of LG Optimus 2X and Samsung Galaxy S5 are obtained from calibration we made. We neglect coefficients p_1, p_2, because their uncertainty is a half of its value or even higher. Furthermore, the coefficients are very small as well as their influence to pixel position. The last equation is transformation from normalize image plane to affine image plane

$$\begin{bmatrix} \mathbf{u}_{rc} \\ 1 \end{bmatrix} \simeq \underline{\mathbf{K}} \begin{bmatrix} \mathbf{u}_d \\ 1 \end{bmatrix},$$

where $\underline{\mathbf{K}}$ is the intrinsic calibration matrix $\underline{\mathbf{K}}_{LG}$, or $\underline{\mathbf{K}}_{samsung}$ that we made in chapter 3.

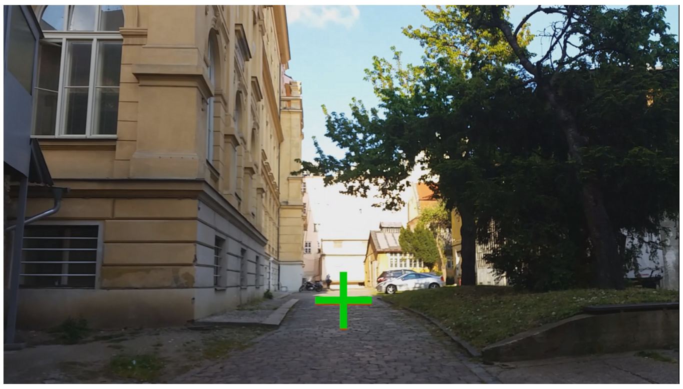
4.1.3 Drawing Marker into the Frame

The last segment of main part takes rounded pixel coordinates $\mathbf{u}_{rc}$ and draws marker in shape of cross on loaded video frame. The loaded frame is three dimensional matrix, where the first dimension is y pixel coordinate, the second dimension is x pixel coordinate and the third dimension is a triplet of RGB values defining a colour of the pixel.

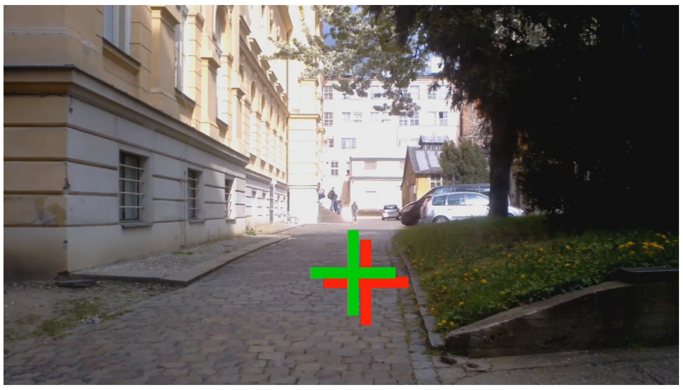
Thus, we draw marker by changing the RGB values of the pixels with x an y coordinates inside the marker shape around the pixel of initial direction. The new RGB values are chosen according to the number of gyroscope sensor in *logfile*.

4.2 Results of Video Augmentation

We made off-line video augmenting tool, which is able to demonstrate ability of gyroscope to track a smartphone rotation. This tool combines our *Naviin* application and our calibration results to do so. The frames from augmented video that we made are shown on figure 4.1.



(a) : Samsung Galaxy S5 - MPU6500 Uncalibrated Gyroscope Sensor, MPU6500 Gyroscope Sensor



(b) : LG Optimus 2X - Corrected Gyroscope Sensor, MPL Gyroscope

Figure 4.1: Complete distortion model of back-facing camera

We can see in image 4.1a, that one marker of Samsung Galaxy S5 covers the another. That correspond with our result from first chapter 2. Thus, we confirmed that the couple of software sensors available in Samsung Galaxy S5 gives us same values, because of the calibration absence.

Chapter 5

Results and Conclusions

The main goal of this work was to determine head orientation for augmented or virtual reality usage. Our approach used IMU already integrated in the devices. This approach preserves comfort of the devices, and allows us to track head orientation at no cost.

We inspected IMU used in contemporary smartphones and found out that many of them already uses sensor fusion to improve IMU characteristics. Thus, we focused on testing already design solutions instead of attempting to design our own. Therefore, we created our own Android application *Naviin*, which captures output sensor data. We chose Android operating system (OS) because it is the OS of many augmented reality products and also it is the OS of the smartphones which can be used in virtual reality device.

We created a tests using a rotation table, which we controlled remotely from our Matlab® script *phoneOnMars* via RS-232. We measured input rotation of the rotation table and response of Samsung Galaxy S5 an LG Optimus 2X fused gyroscope. Then we created Matlab function® to load all measured data and plot selected one, with possibility to integrate them once or twice before plotting.

From the results, we found out, that Samsung Galaxy S5 did not have calibrated initial zero-rate offset and so significantly drifted. The LG Optimus 2X gyroscope data suffered by error with unknown origin, which made several values many times bigger than input rotation. We proposed the solution, which was substitution of corrupt values with average of previous and following values. Then we tested proposed solution and found out that solution improved the response to input rotation. Furthermore, the results even seemed to suppress the drift.

After this discovery, we continued with camera calibration in order to get intrinsic calibration matrices of the Samsung Galaxy S5 and LG Optimus 2X back-facing camera. We used Camera Calibration Toolbox for Matlab® and made intrinsic calibration matrices and distortion parameters. The resulting coefficients of tangential distortions turned out to be small with big uncertainties (approximately three times the standard deviations), which were in some cases even bigger than the coefficient itself. Therefore, we neglected them and used a radial distortion only. The radial distortion of newer Samsung Galaxy S5 had significantly bigger radial distortion than LG Optimus 2X.

In the and, we utilized previous effort and created Matlab® function *gyroToVideo*. The *gyroToVideo* used both our *Naviin* application and our intrinsic calibration matrices with radial distortions. The *Naviin* was used to record video and capture output sensor data, while intrinsic calibration matrices were used to display direction of an initial camera facing to the video frames. Then, we inspected how much fixed is the initial direction marker to the context of video. We found out, that both Samsung Galaxy S5 and LG Optimus 2X markers drifted. That could have been caused by the fact, that the context was not as far as the *direction point*, which was in infinity. Consequently, the translation of camera coordinate system might have caused the markers to drift.

The conclusions of this work are, that embedded sensor fusion algorithm in smartphones does not seems to be as potential as expected. Especially when manufacturer does not calibrate the sensors. Therefore an additional software sensor fusion or data filtering is necessary in order to obtain results, which would be possible to use in AR glasses or VR devices.

In future, it would be great to test, if the translation of camera coordinate system is really causing the drift. The way to do so, it is usage of a fixed Cardan suspension, which would enabled orientation change without position change.

Also implementing video editing on-line in smartphone could give us interesting results. The results concerning an important power efficiency. When power efficiency is tought in either computation power and power management purpose.

Abbreviations

AMR	Anisotropic Magnetoresistance
API	Aplication Program Interface
AR	Augmented Reality
CSV	Comma-Separated Values
DoF	Degree of Freedom
IMU	Inertial measurement unit
GPS	Global Positioning System
GMR	Giant Magnetoresistance
GNSS	Global Navigation Satellite System
GLONASS	Globalnaya navigatsionnaya sputnikovaya sistema
MEMS	Micro-electro-mechanical system
MI	Magnetoimpedance
MVG	Micromachined Vibratory Gyroscopes
OS	Operating System
VR	Virtual Reality

Bibliography

[1] http://www.gartner.com/newsroom/id/3215217.

[2] Kaveh Mohamadabadi. Anisotropic Magnetoresistance Magnetometer for inertial navigation systems. Electronics. Ecole Polytechnique X, 2013. English. <tel-00946970>.

[3] http://developer.android.com/reference/android/hardware/SensorEvent.html.

[4] http://cmp.felk.cvut.cz/~pisa/mars/mars_man_en.pdf.

[5] http://www.vision.caltech.edu/bouguetj/calib_doc/index.html#ref.

[6] Aichi steel ami304 specification. Ver.B9110329 http://sd652abba6b419853.jimcontent.com/download/version/1459919721/module/9577378992/name/b9110329ami304e.pdf.

[7] Asahi kasei microsystems ak09911 3-axis electronic compass. www.akm.com/akm/en/file/datasheet/AK09911.pdf.

[8] Kionix kxtf9-1026 tri-axis digital accelerometer specifications, 1 2011. Rev. 6 http://kionixfs.kionix.com/en/datasheet/KXTF9-1026%20Specifications%20Rev%206.pdf.

[9] Mpu-3050 motion processing unit product specification, 11 2011. Rev. 2.9 https://store.invensense.com/datasheets/invensense/MPU-3000A.pdf.

[10] Mpu-6500 product specification, 9 2013. Rev. 1.0 https://store.invensense.com/datasheets/invensense/MPU_6500_Rev1.0.pdf.

[11] P. Aggarwal. *MEMS-based integrated navigation.* Artech House, Boston, 2010. ISBN 978-1-60807-043-5.

[12] M. Bednařík. *Fyzika 1.* České vysoké učení technické, V Praze, first edition, 2011. ISBN 978-80-01-04834-4. 63-67 pp.

[13] S. Bhattacharyya. *Handbook of signal processing systems.* Springer, New York, NY, 2013. ISBN 978-1-4614-6858-5.

[14] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[15] R. P. G. Collinson. *Introduction to avionics systems.* Kluwer Academic, Boston London, 2003. ISBN 978-1-4419-7466-2.

[16] J. Heikkila and O. Silvén. A four-step camera calibration procedure with implicit image correction. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1106–1112. IEEE, 1997.

[17] V. Hlaváč. ae3m33iro lecture 02: Geometry of a single camera. lecture presented at, 2015. CTU in Prague.

[18] Y. Honkura, M. Yamamoto, N. Hamada, and A. Shimode. Ultra-sensitive magnetoimpedance sensor, 2 2012. Patent application, EP 2423697 A1.

[19] R. J. Radke. *Computer vision for visual effects.* Cambridge University Press, 2012.

[20] E. Ramsden. *Hall-effect sensors : theory and applications.* Elsevier/Newnes, Amsterdam Boston, 2006. ISBN 978-0-7506-7934-3.

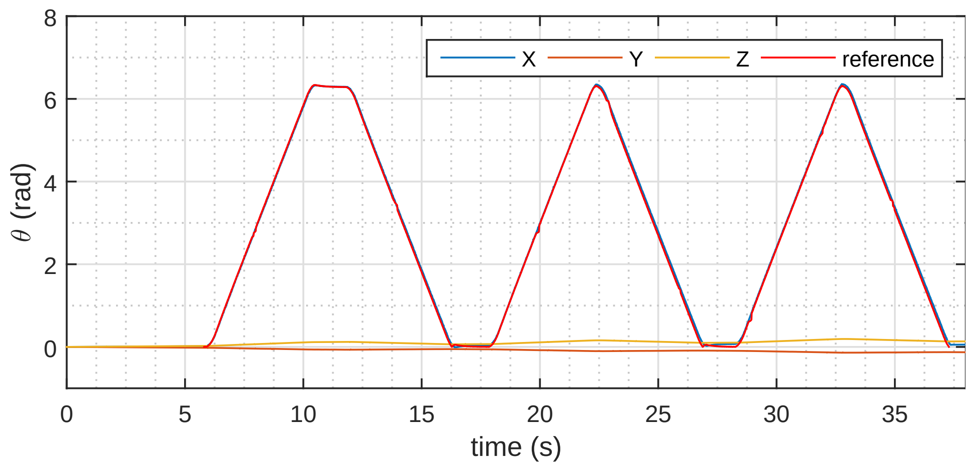
[21] C. Reig. *Giant magnetoresistance (GMR) sensors from basis to state-of-the-art applications.* Springer, Berlin New York, 2013. ISBN 978-3-642-37172-1.

[22] A. M. Shkel. Type i and type ii micromachined vibratory gyroscopes. In *Position, Location, and Navigation Symposium, San Diego*, pages 25–27, 2006.

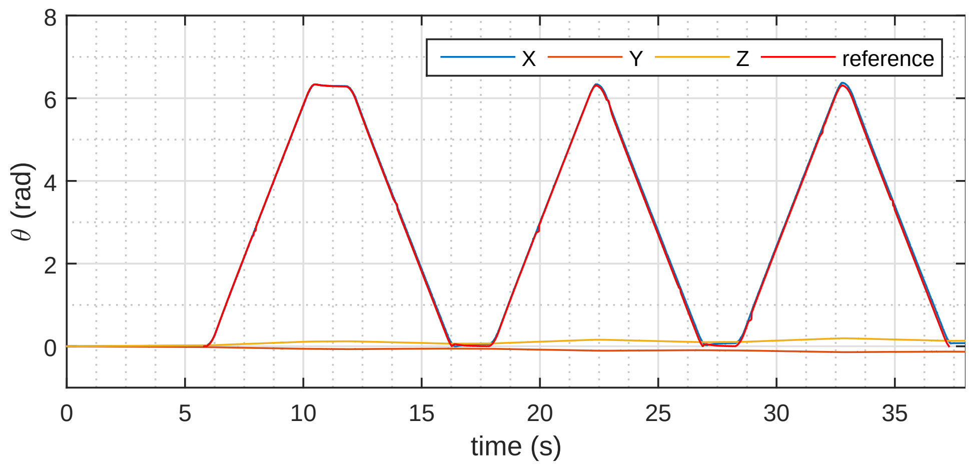
[23] P. L. Walter. The history of the accelerometer: 1920s-1996-prologue and epilogue, 2006. *Sound & vibration*, 41(1):84–90, 2007.

[24] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 666–673. IEEE, 1999.

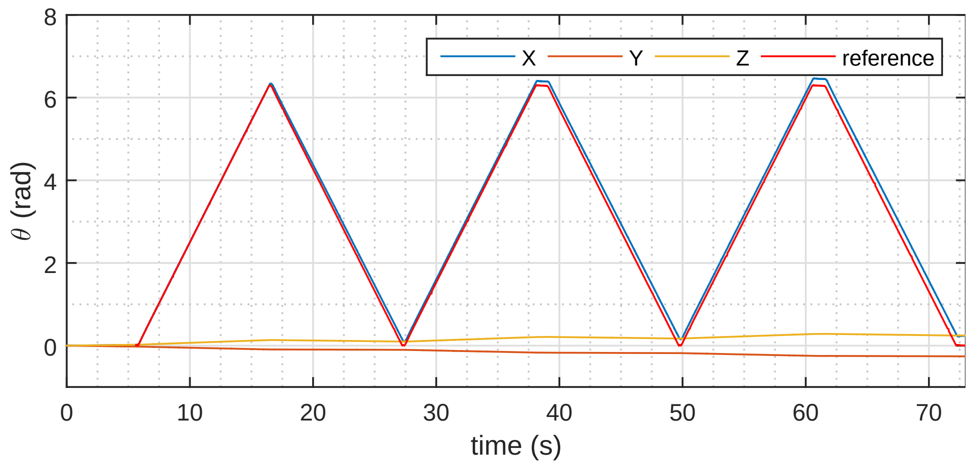
Appendix A
Samsung Galaxy S5 All Measurements



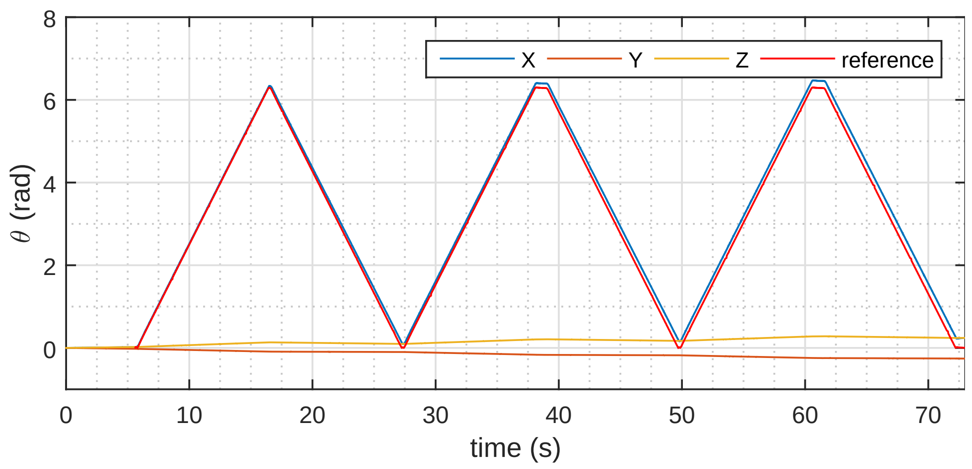
(a) : $M_S = 1000$ - MPU6500 Gyroscope Sensor



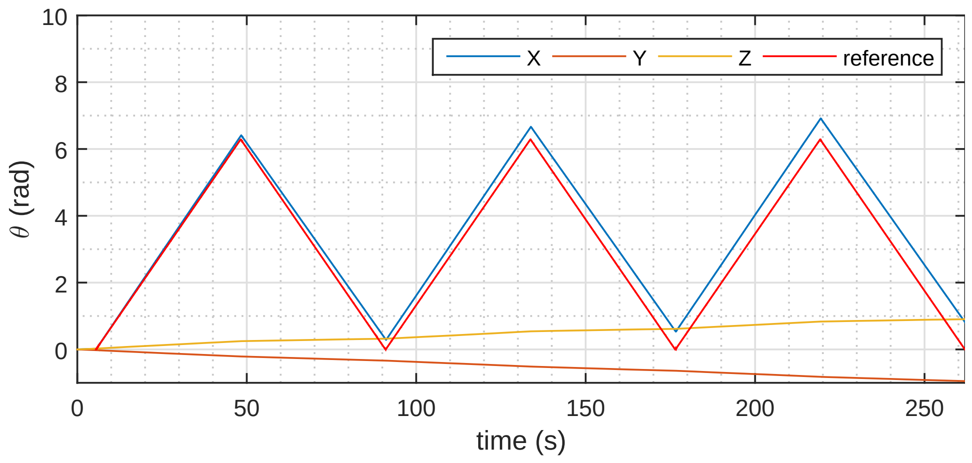
(b) : $M_S = 1000$ - MPU6500 Uncalibrated Gyroscope Sensor



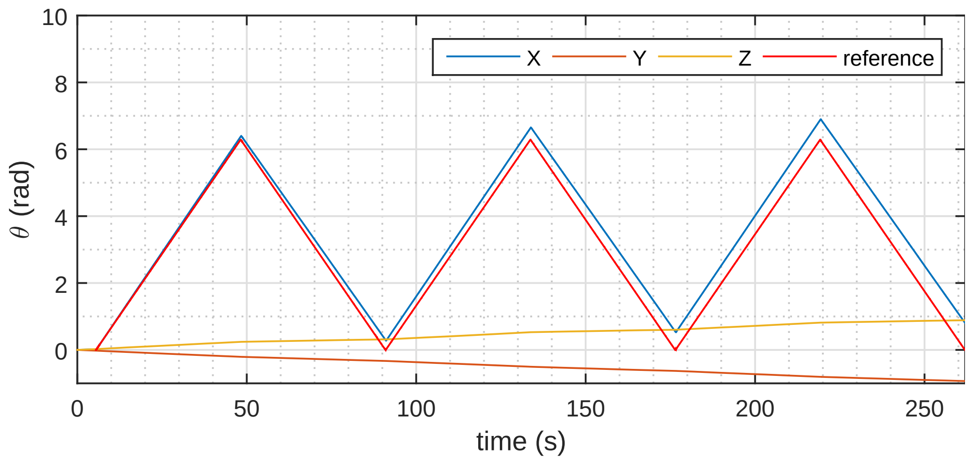
(c) : $M_S = 400$ - MPU6500 Gyroscope Sensor



(d) : $M_S = 400$ - MPU6500 Uncalibrated Gyroscope Sensor

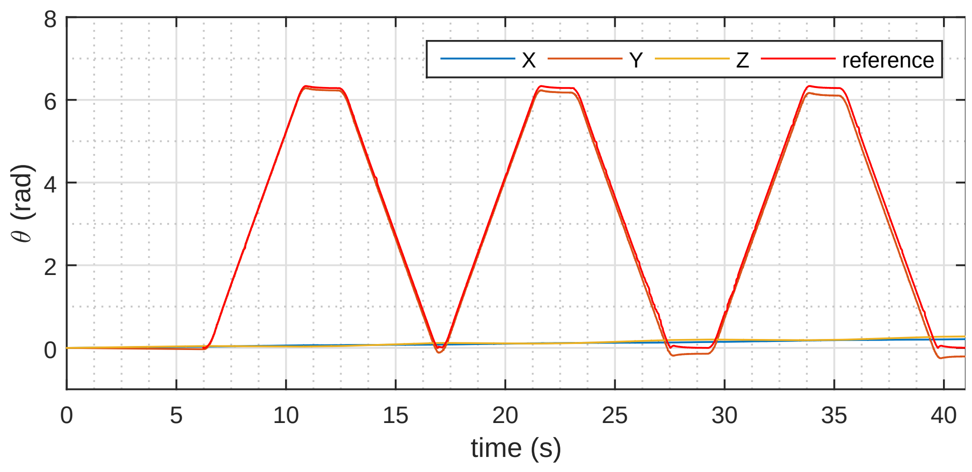


(e) : $M_S = 100$ - MPU6500 Gyroscope Sensor

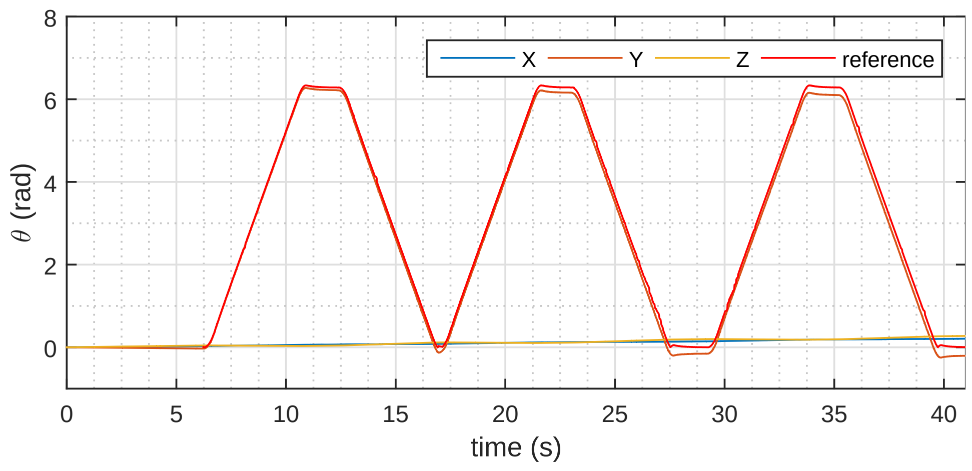


(f) : $M_S = 100$ - MPU6500 Uncalibrated Gyroscope Sensor

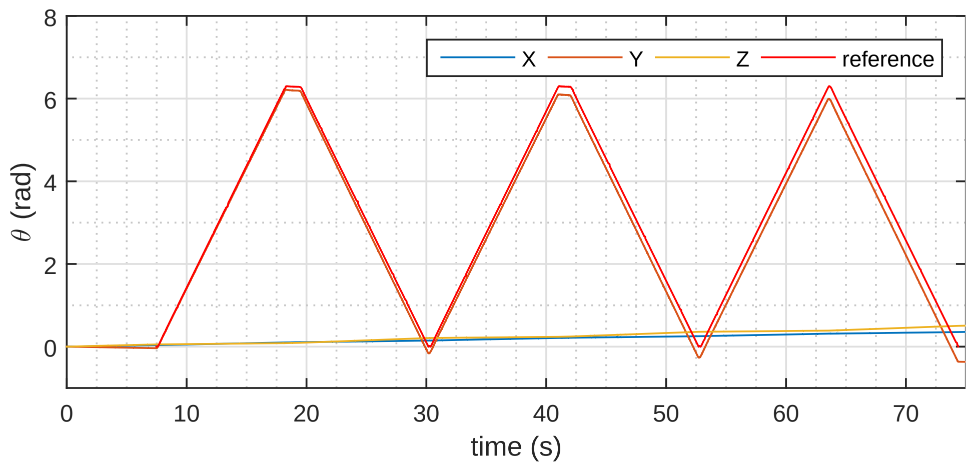
Figure A.1: Samsung x-axis rotation test



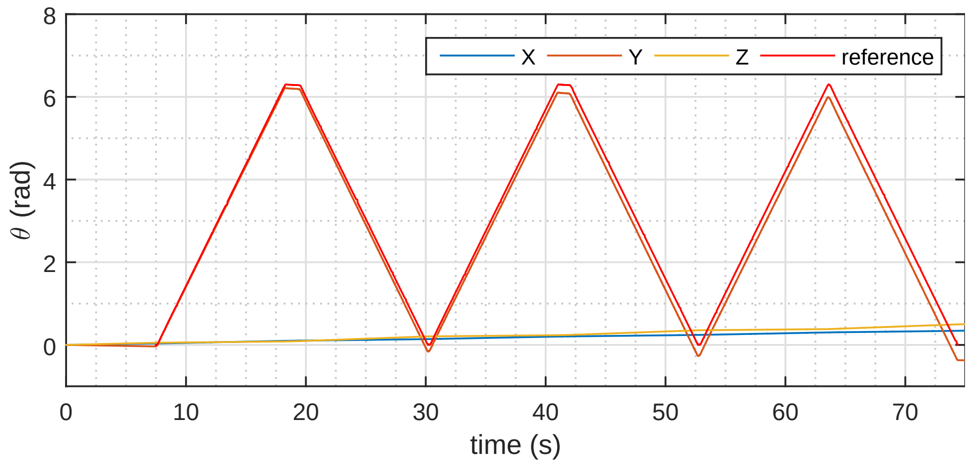
(a) : $M_S = 1000$ - MPU6500 Gyroscope Sensor



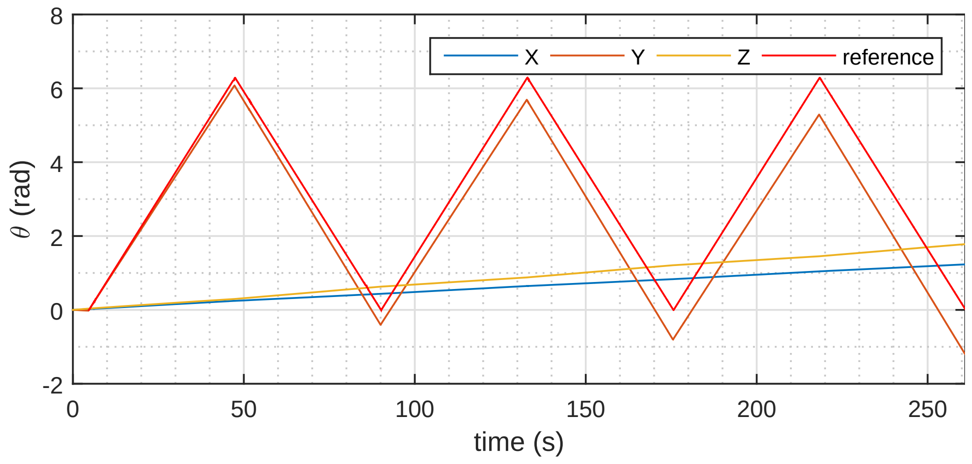
(b) : $M_S = 1000$ - MPU6500 Uncalibrated Gyroscope Sensor



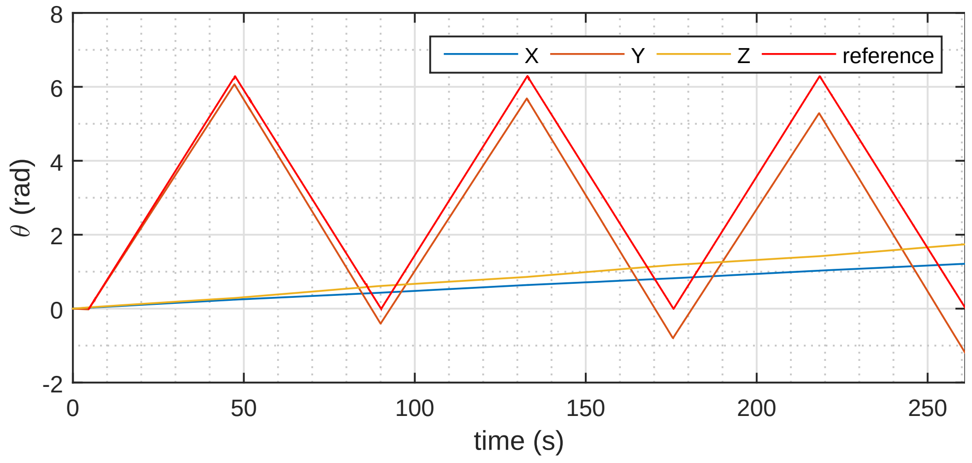
(c) : $M_S = 400$ - MPU6500 Gyroscope Sensor



(d) : $M_S = 400$ - MPU6500 Uncalibrated Gyroscope Sensor

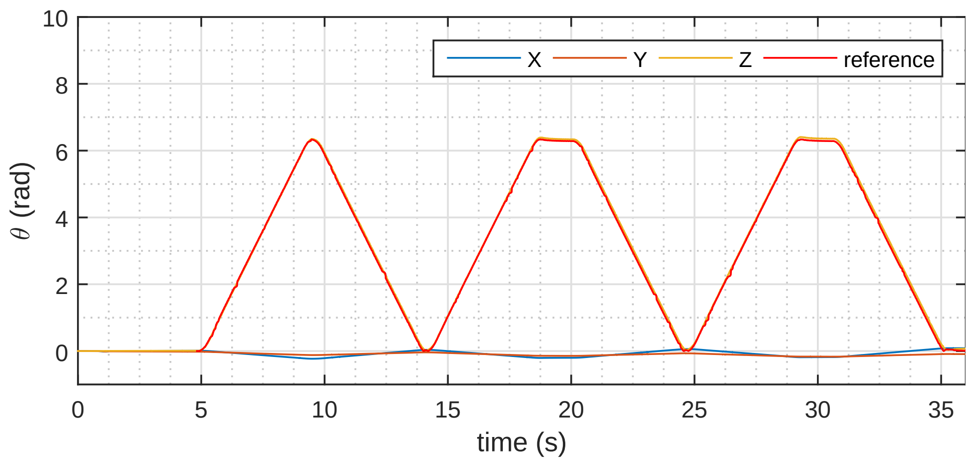


(e) : $M_S = 100$ - MPU6500 Gyroscope Sensor

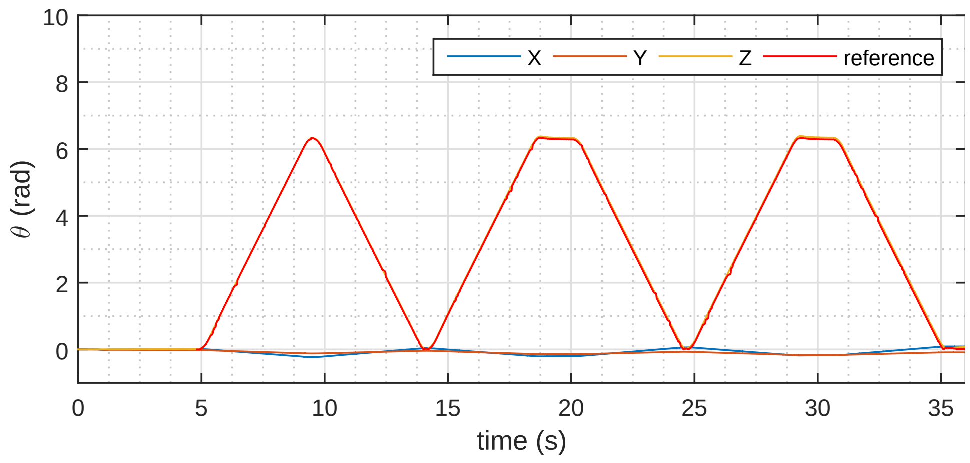


(f) : $M_S = 100$ - MPU6500 Uncalibrated Gyroscope Sensor

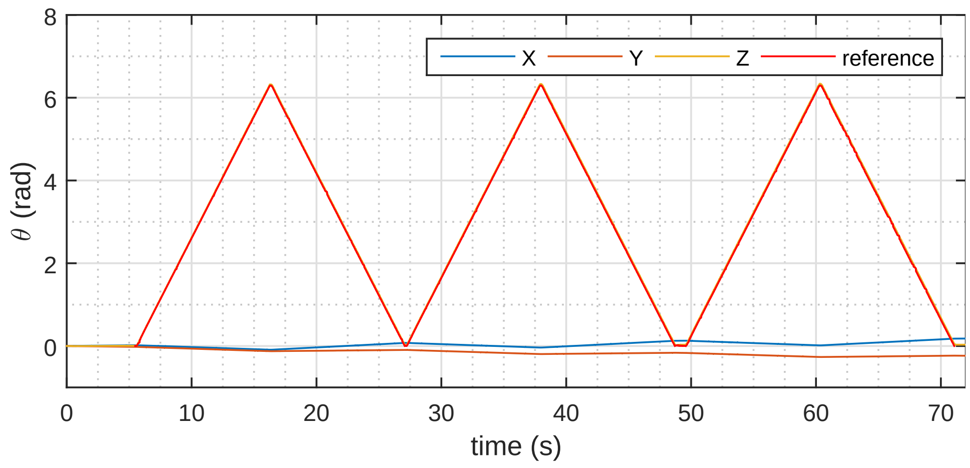
Figure A.2: Samsung y-axis rotation test



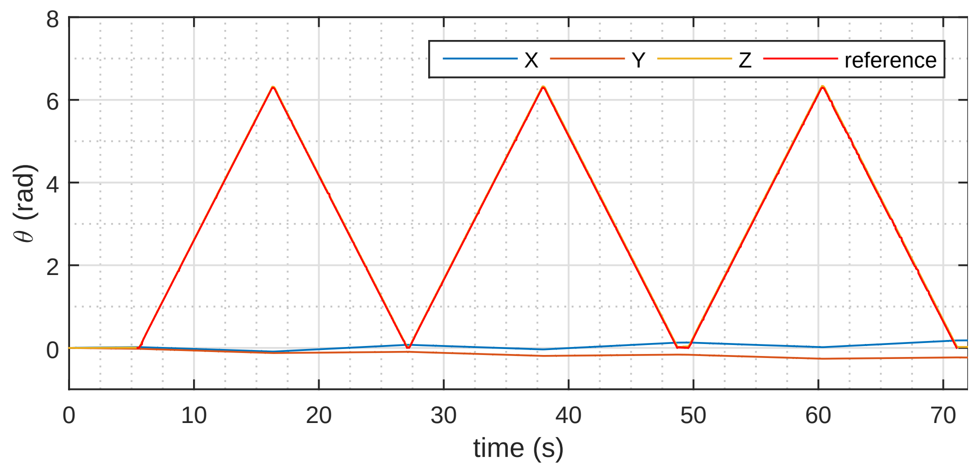
(a) : $M_S = 1000$ - MPU6500 Gyroscope Sensor



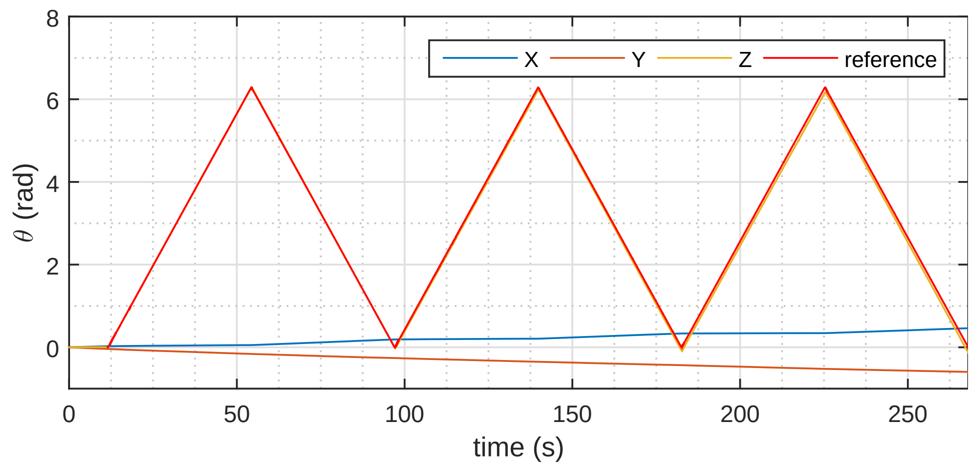
(b) : $M_S = 1000$ - MPU6500 Uncalibrated Gyroscope Sensor



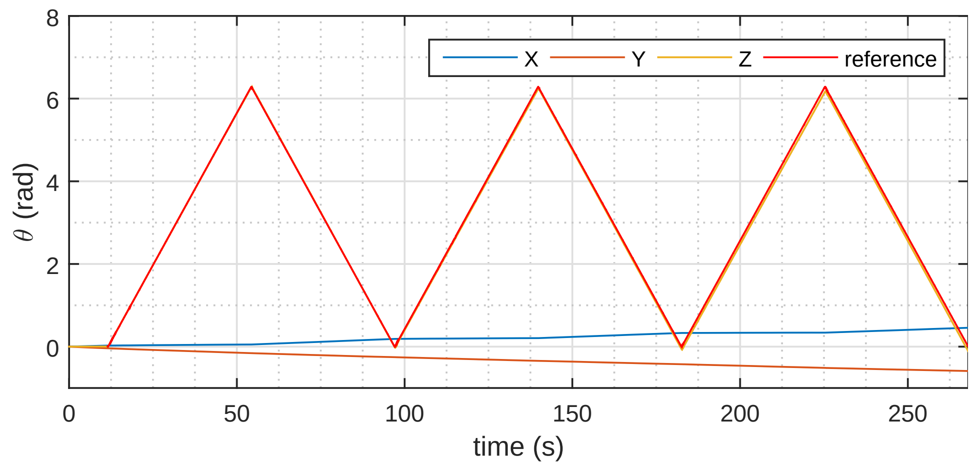
(c) : $M_S = 400$ - MPU6500 Gyroscope Sensor



(d) : $M_S = 400$ - MPU6500 Uncalibrated Gyroscope Sensor



(e) : $M_S = 100$ - MPU6500 Gyroscope Sensor

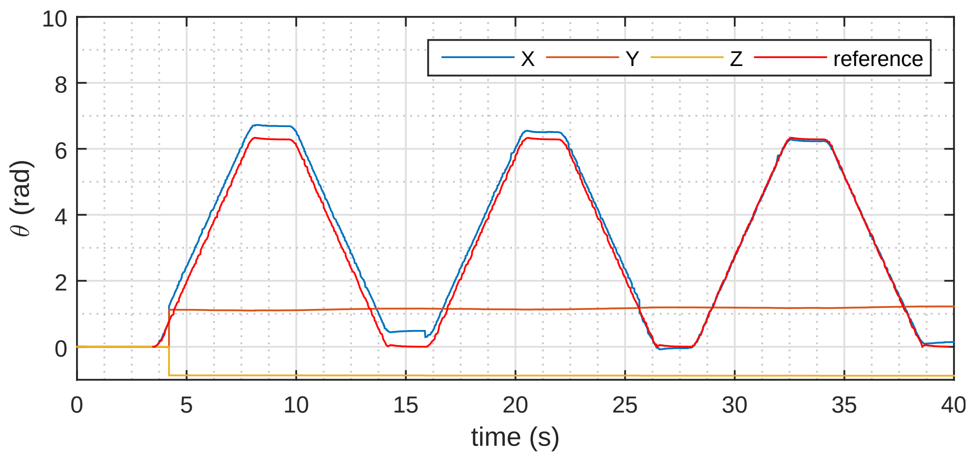


(f) : $M_S = 100$ - MPU6500 Uncalibrated Gyroscope Sensor

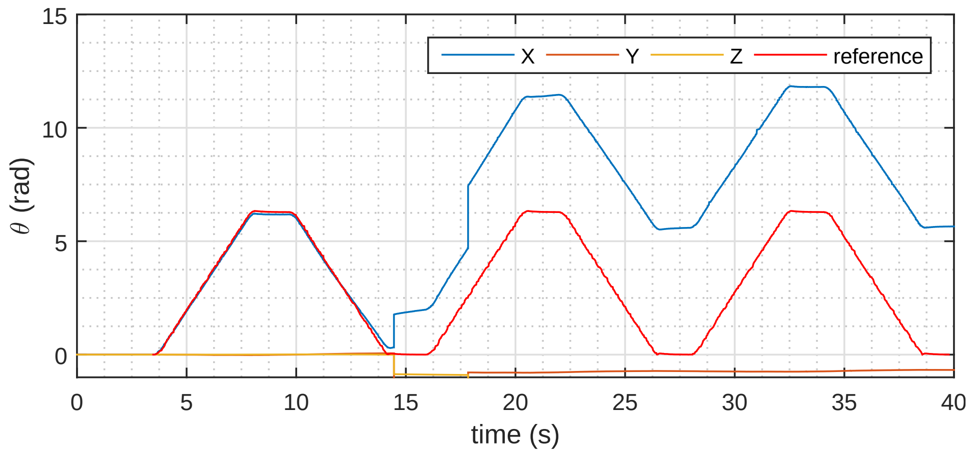
Figure A.3: Samsung z-axis rotation test

Appendix B

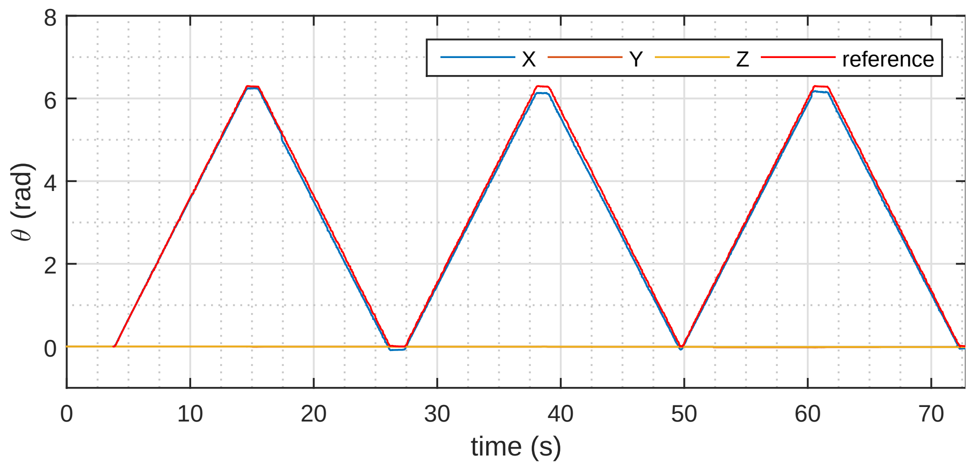
LG Optimus 2X All Measurements



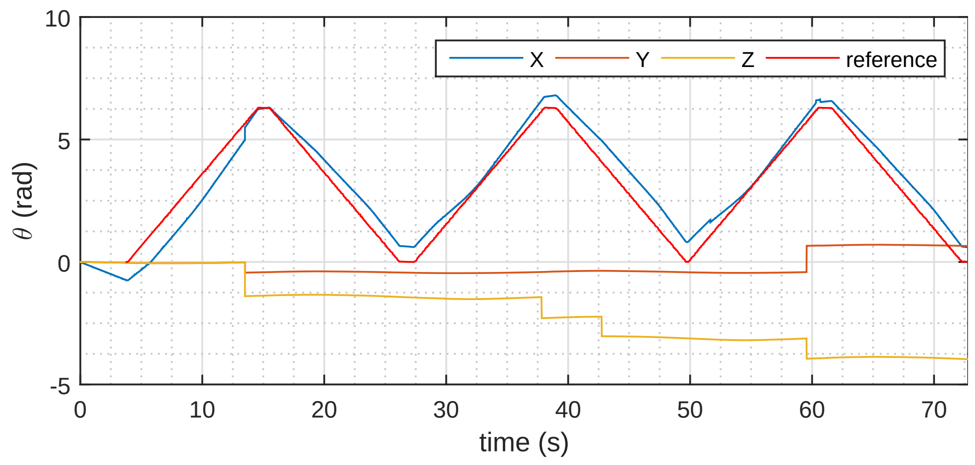
(a) : $M_S = 1000$ - MPL Gyroscope



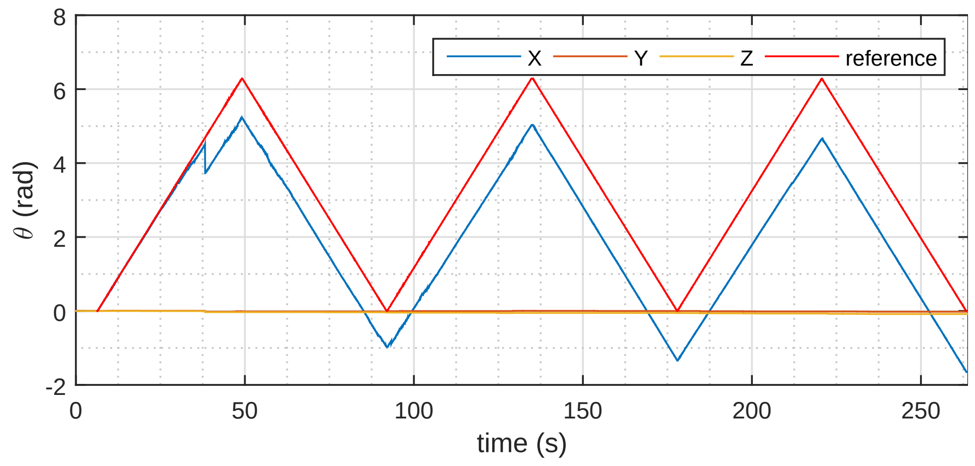
(b) : $M_S = 1000$ - Corrected Gyroscope Sensor



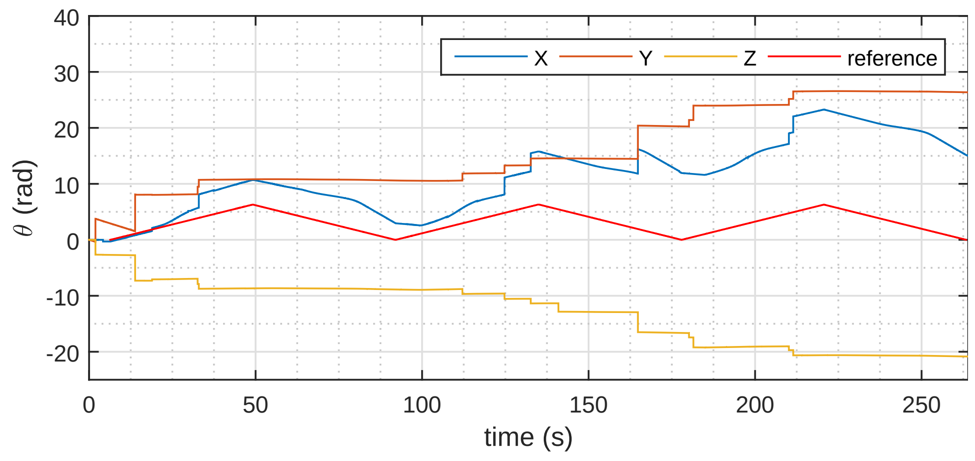
(c) : $M_S = 400$ - MPL Gyroscope



(d) : $M_S = 400$ - Corrected Gyroscope Sensor

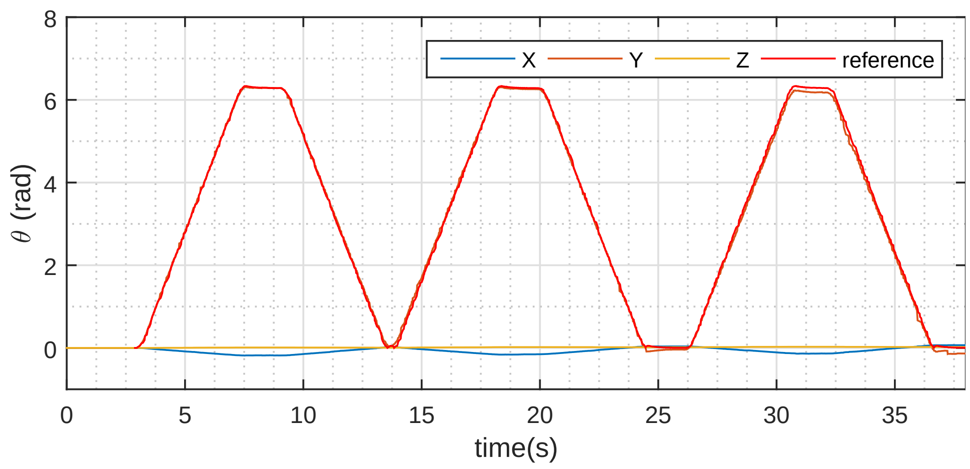


(e) : $M_S = 100$ - MPL Gyroscope

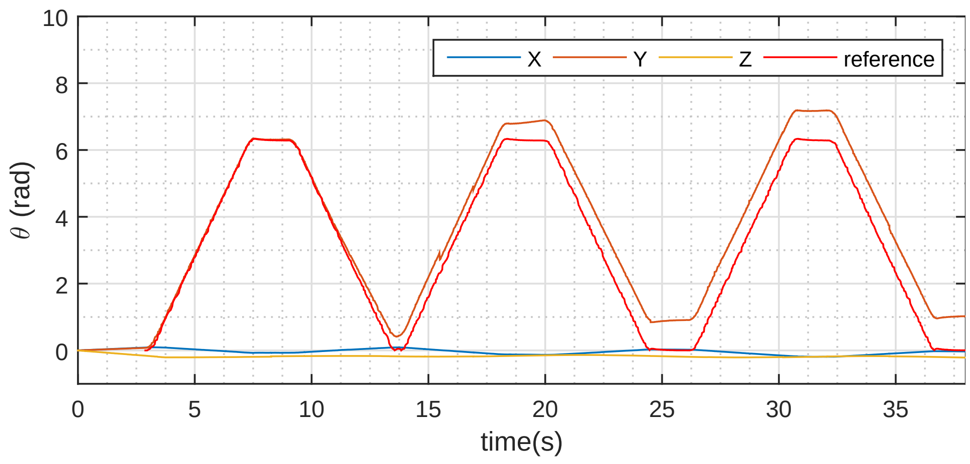


(f) : $M_S = 100$ - Corrected Gyroscope Sensor

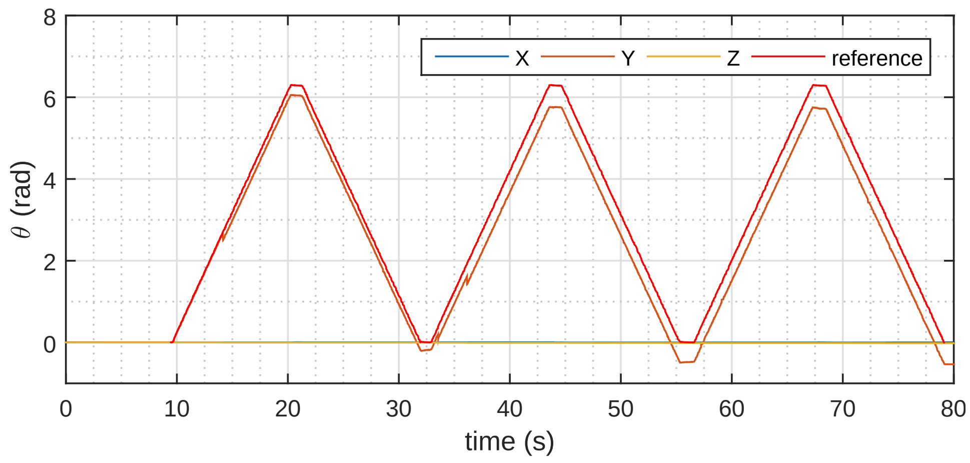
Figure B.1: LG x-axis rotation test



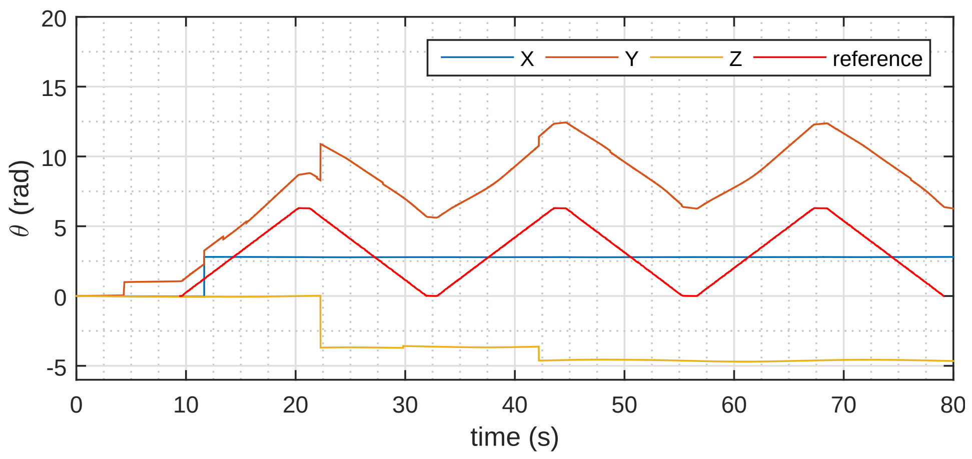
(a) : $M_S = 1000$ - MPL Gyroscope



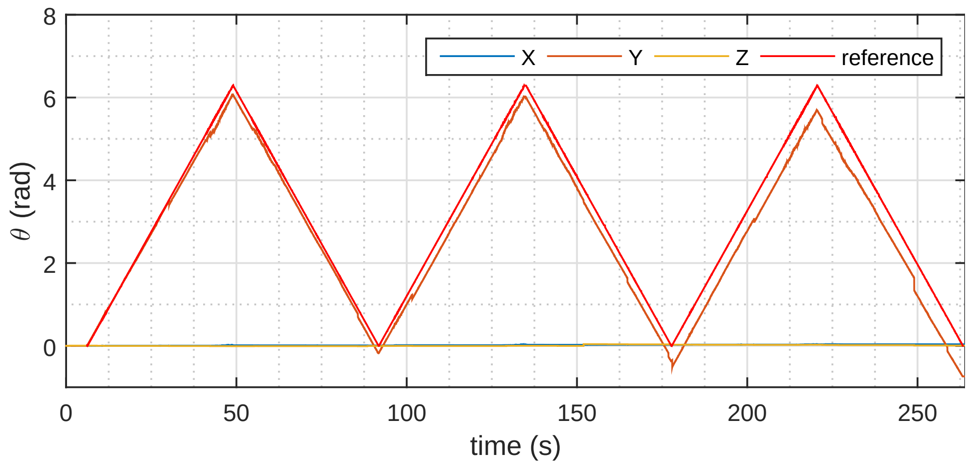
(b) : $M_S = 1000$ - Corrected Gyroscope Sensor



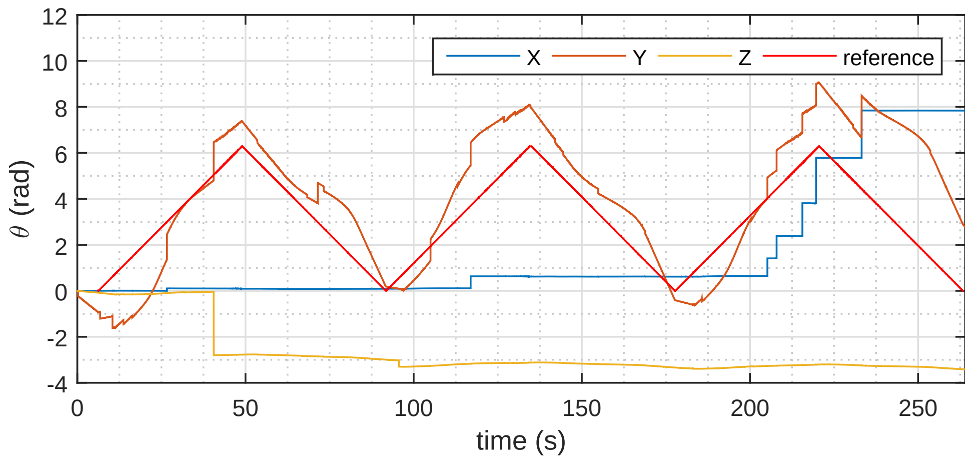
(c) : $M_S = 400$ - MPL Gyroscope



(d) : $M_S = 400$ - Corrected Gyroscope Sensor

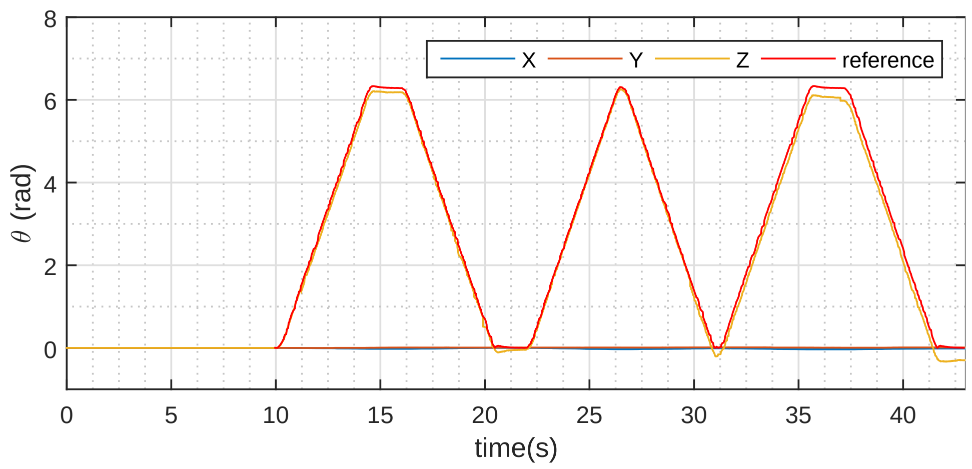


(e) : $M_S = 100$ - MPL Gyroscope

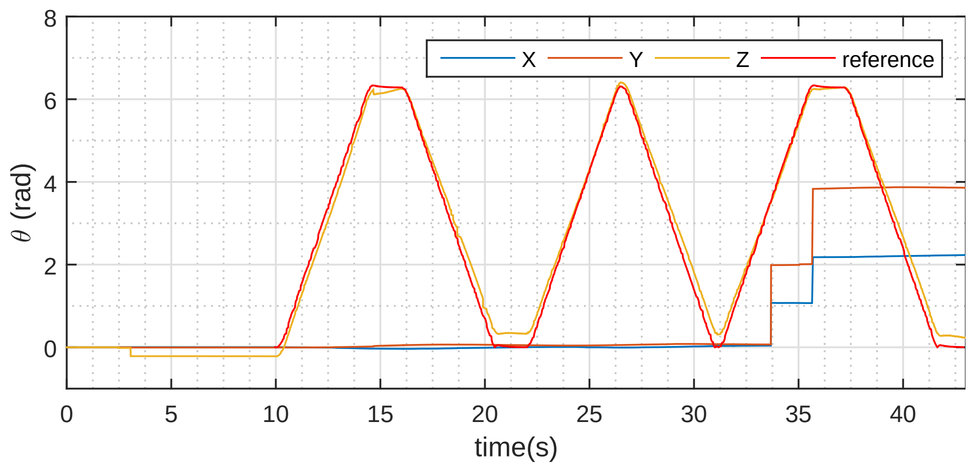


(f) : $M_S = 100$ - Corrected Gyroscope Sensor

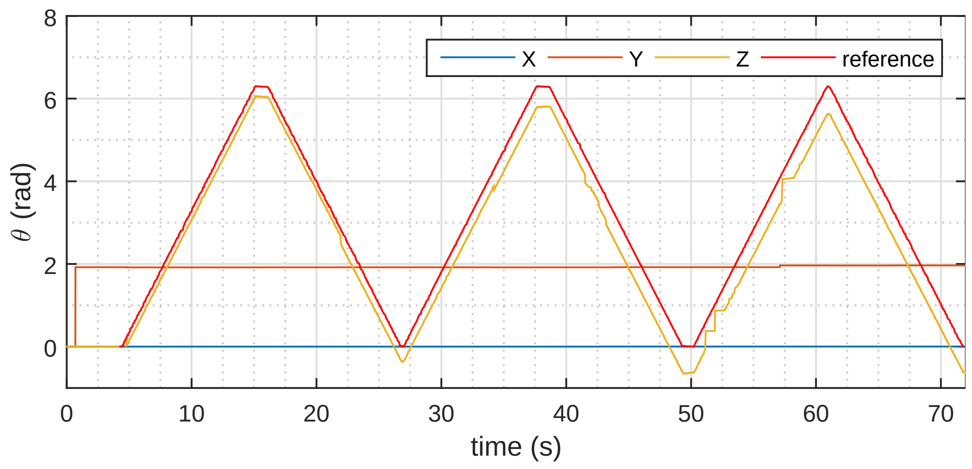
Figure B.2: LG y-axis rotation test



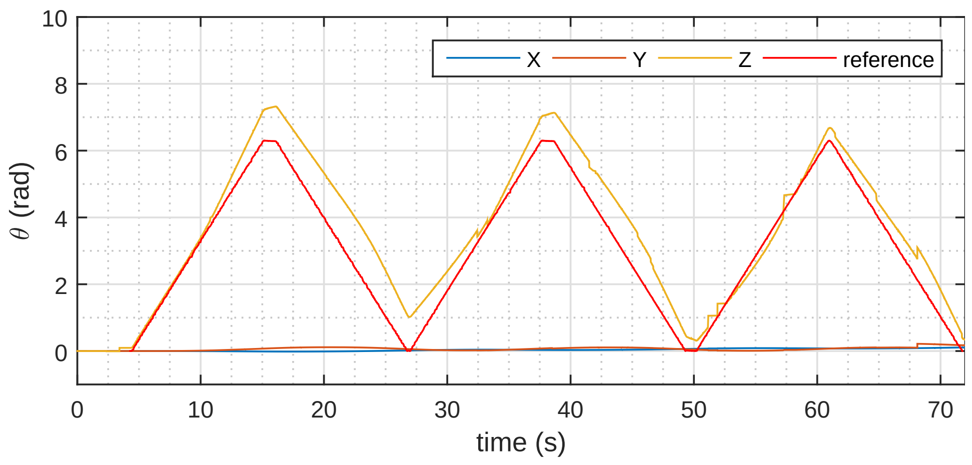
(a) : $M_S = 1000$ - MPL Gyroscope



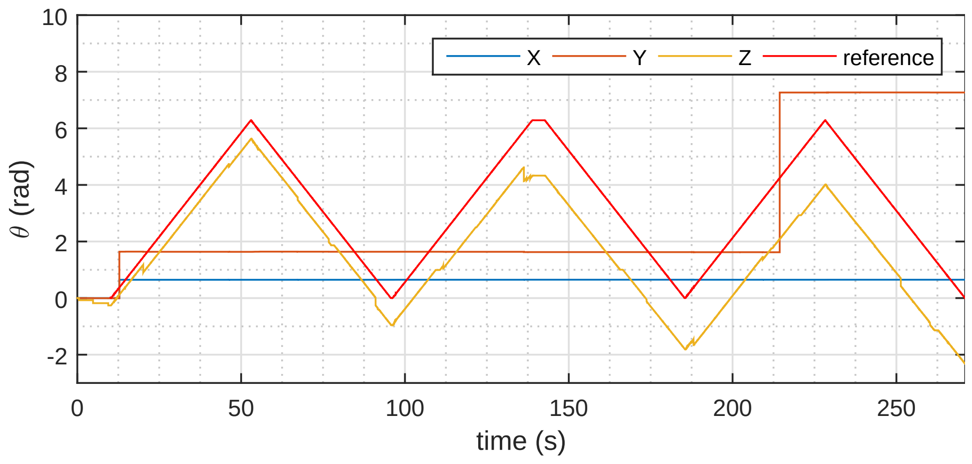
(b) : $M_S = 1000$ - Corrected Gyroscope Sensor



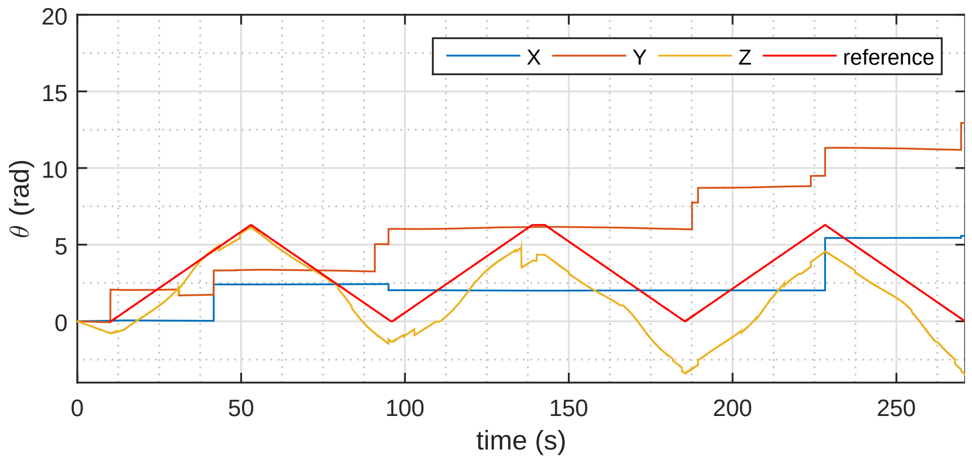
(c) : $M_S = 400$ - MPL Gyroscope



(d) : $M_S = 400$ - Corrected Gyroscope Sensor



(e) : $M_S = 100$ - MPL Gyroscope

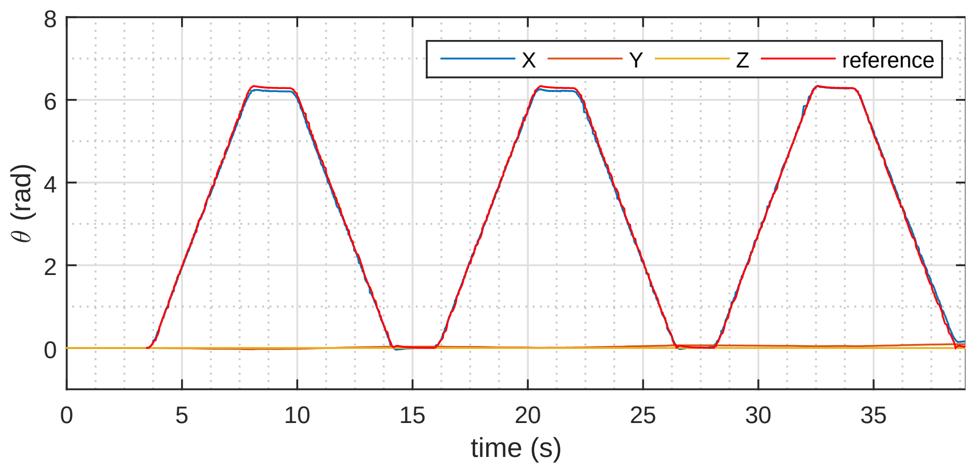


(f) : $M_S = 100$ - Corrected Gyroscope Sensor

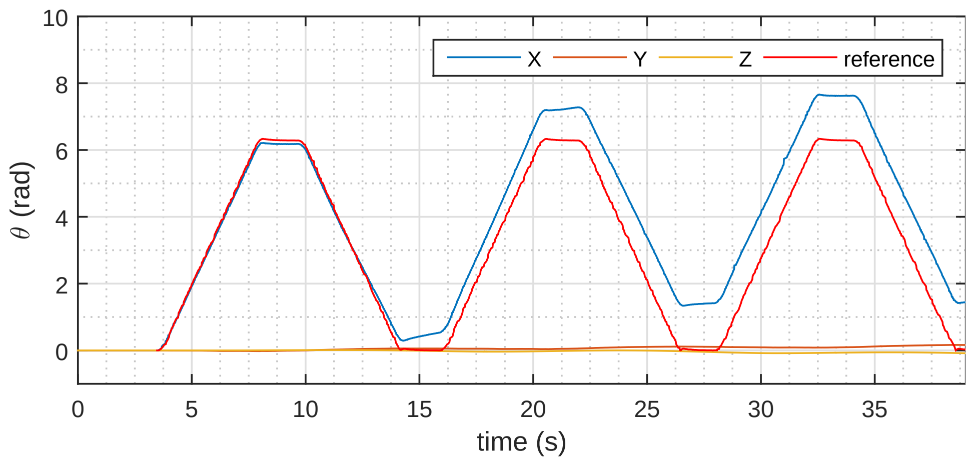
Figure B.3: LG z-axis rotation test

Appendix C

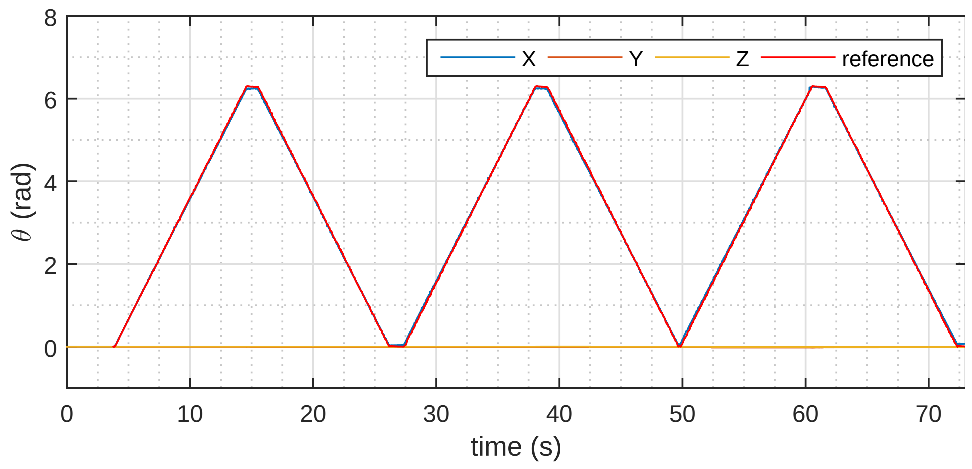
LG Optimus 2X All Measurements After Algorithm Improvement



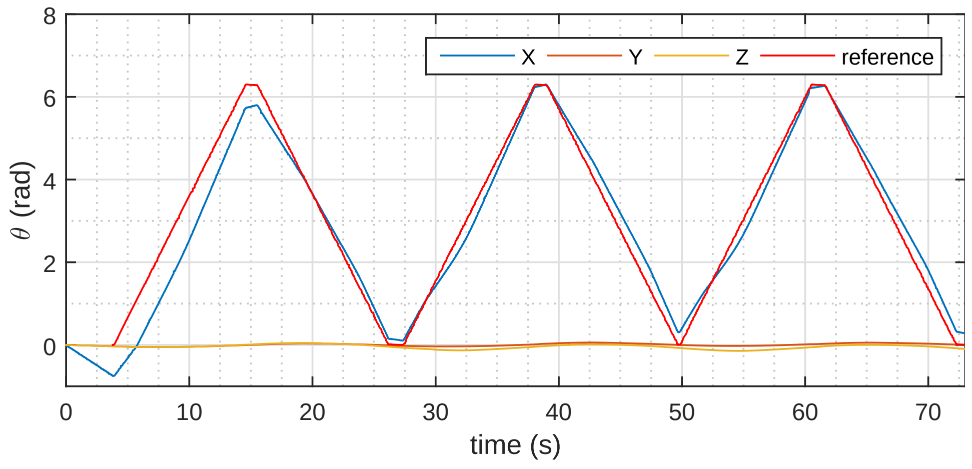
(a) : $M_S = 1000$ - MPL Gyroscope



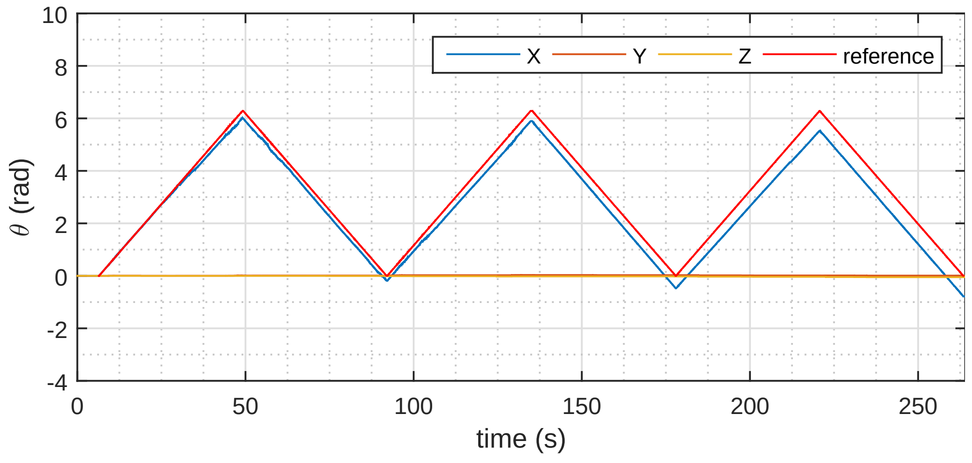
(b) : $M_S = 1000$ - Corrected Gyroscope Sensor



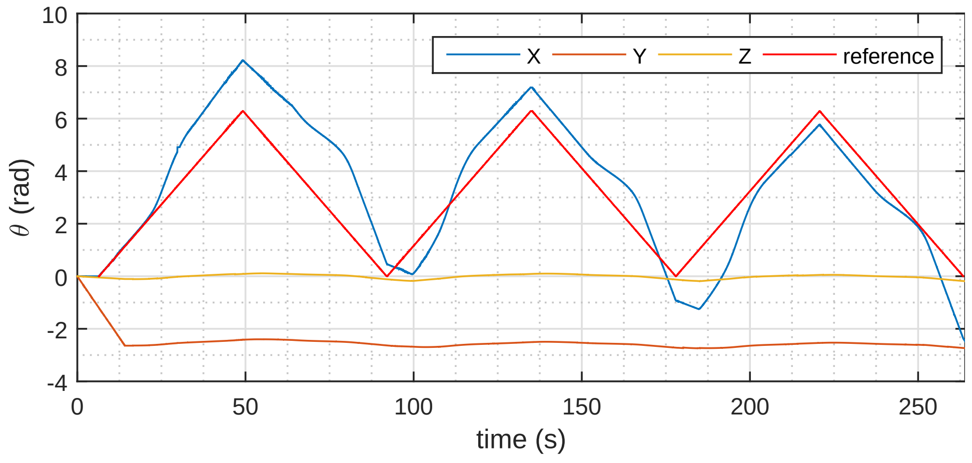
(c) : $M_S = 400$ - MPL Gyroscope



(d) : $M_S = 400$ - Corrected Gyroscope Sensor

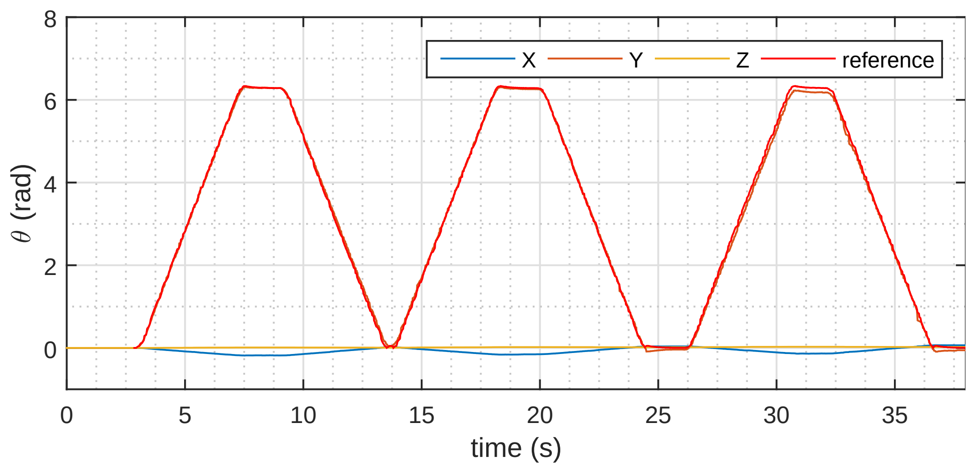


(e) : $M_S = 100$ - MPL Gyroscope

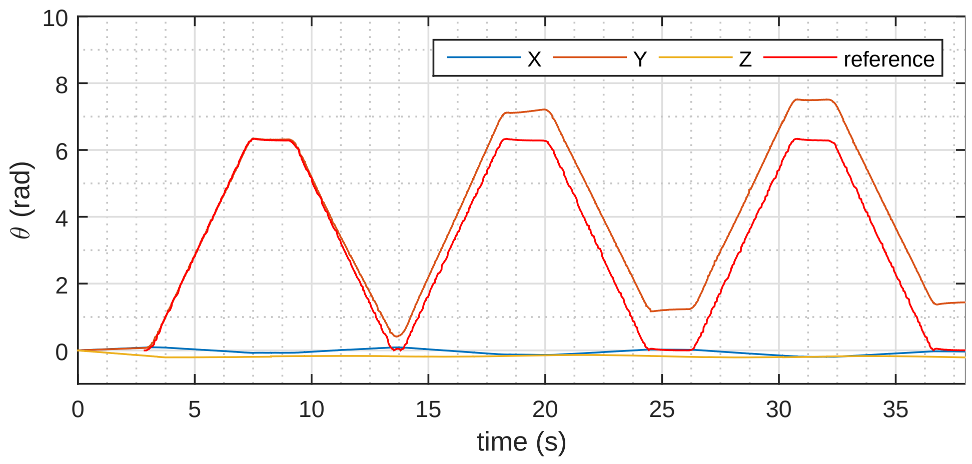


(f) : $M_S = 100$ - Corrected Gyroscope Sensor

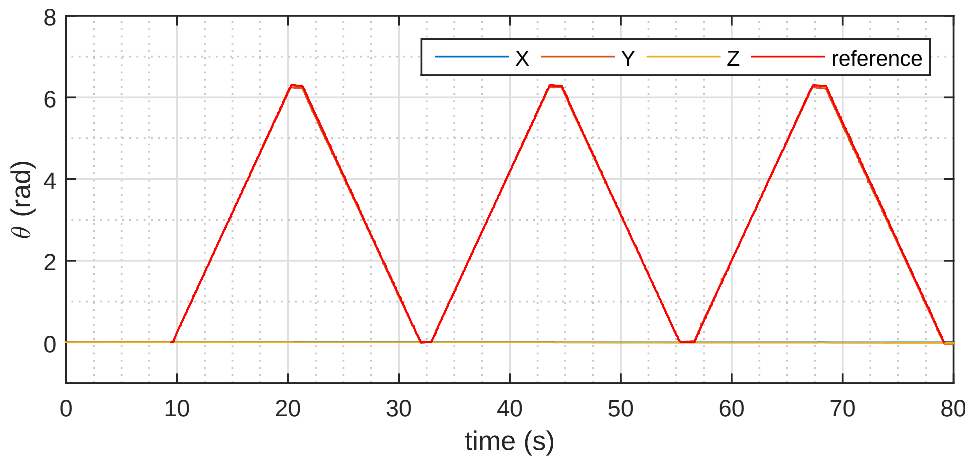
Figure C.1: LG x-axis rotation test with filter



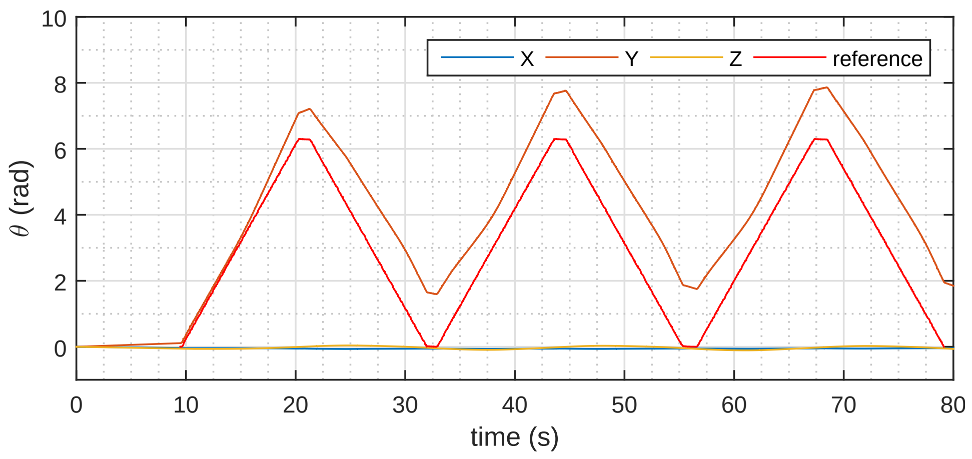
(a) : $M_S = 1000$ - MPL Gyroscope



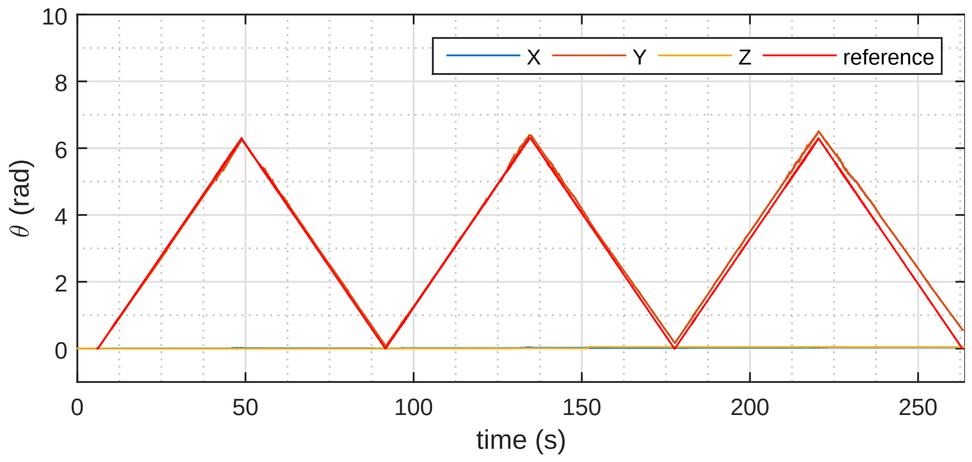
(b) : $M_S = 1000$ - Corrected Gyroscope Sensor



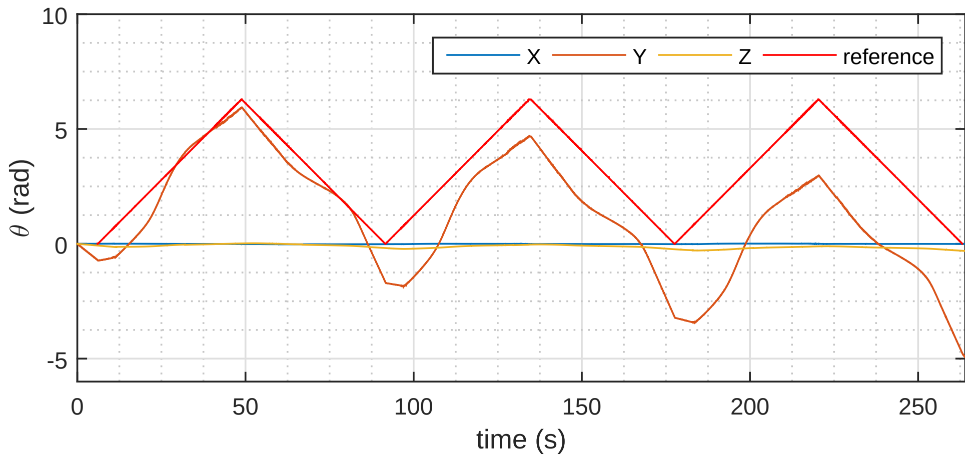
(c) : $M_S = 400$ - MPL Gyroscope



(d) : $M_S = 400$ - Corrected Gyroscope Sensor

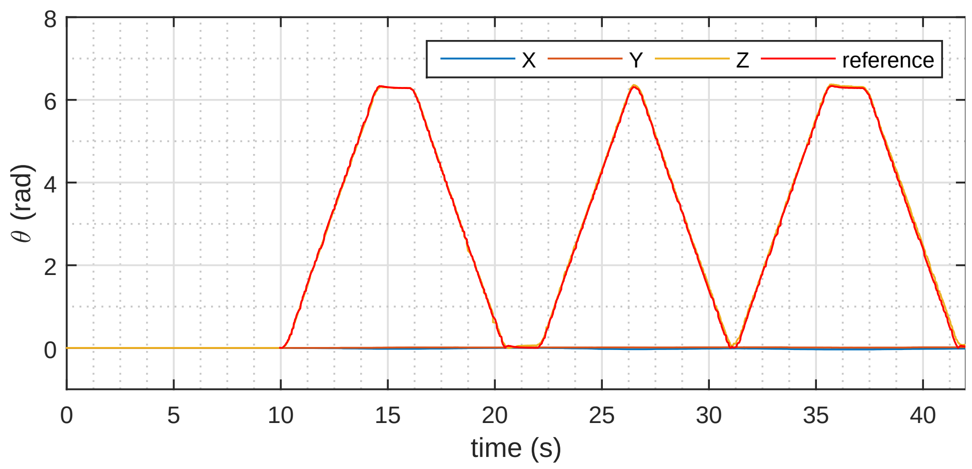


(e) : $M_S = 100$ - MPL Gyroscope

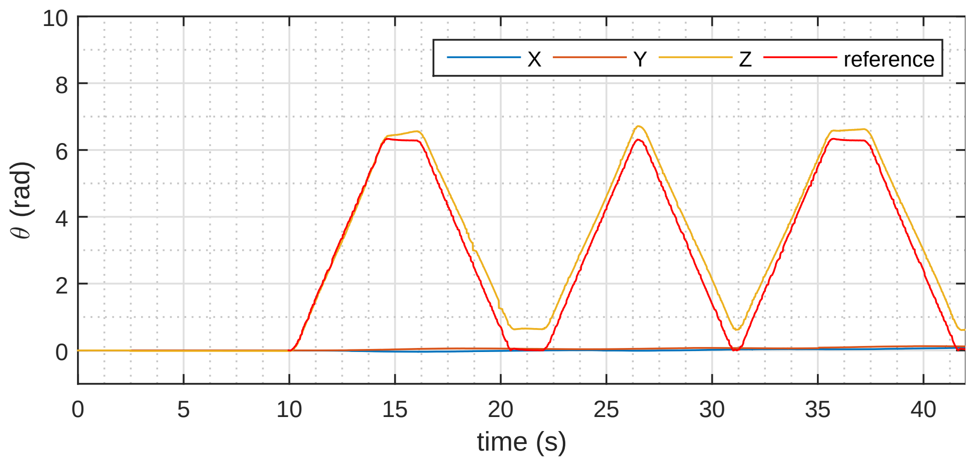


(f) : $M_S = 100$ - Corrected Gyroscope Sensor

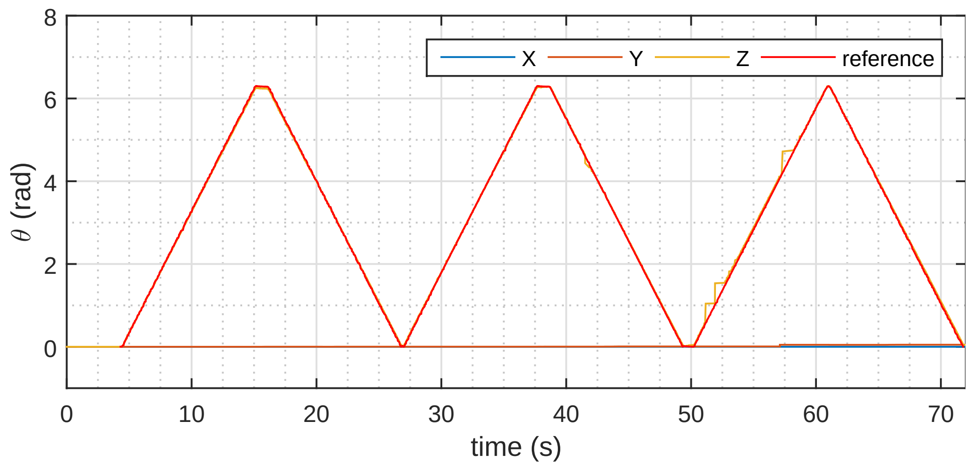
Figure C.2: LG y-axis rotation test with filter



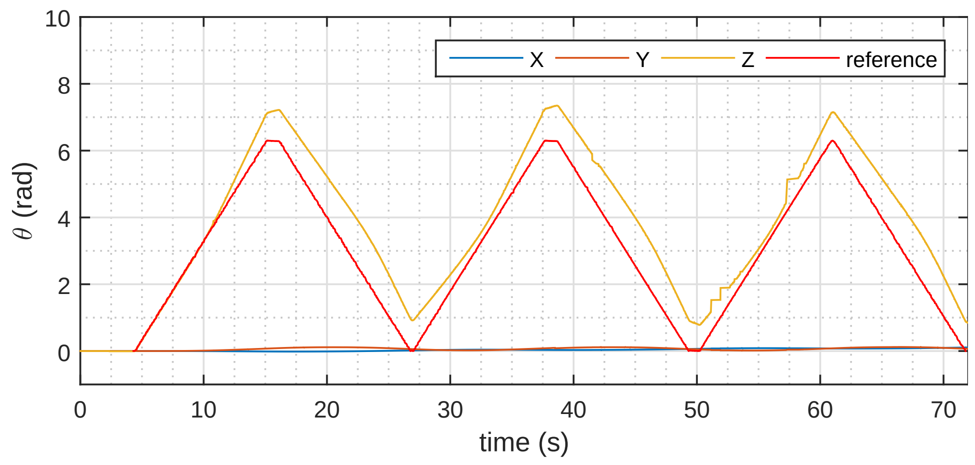
(a) : $M_S = 1000$ - MPL Gyroscope



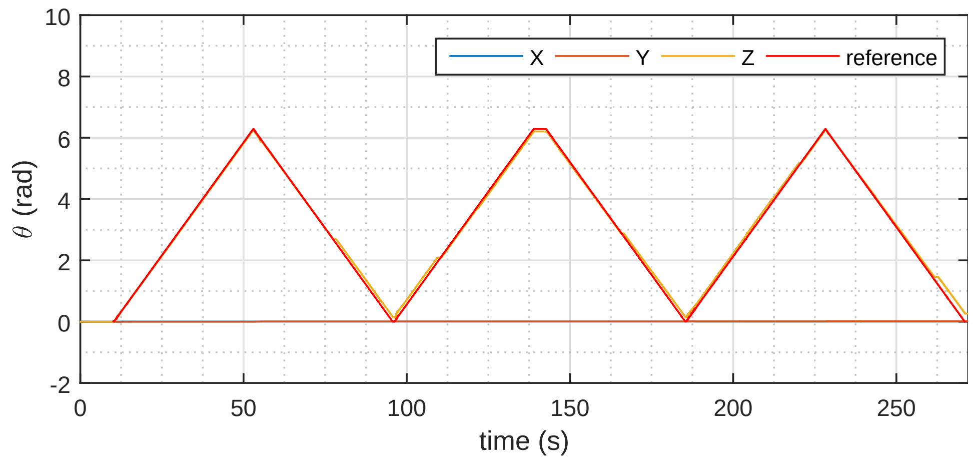
(b) : $M_S = 1000$ - Corrected Gyroscope Sensor



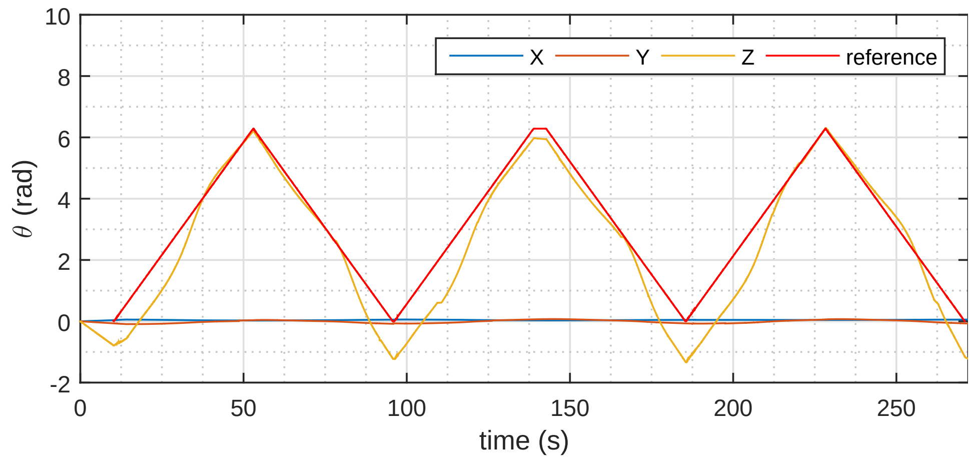
(c) : $M_S = 400$ - MPL Gyroscope



(d) : $M_S = 400$ - Corrected Gyroscope Sensor



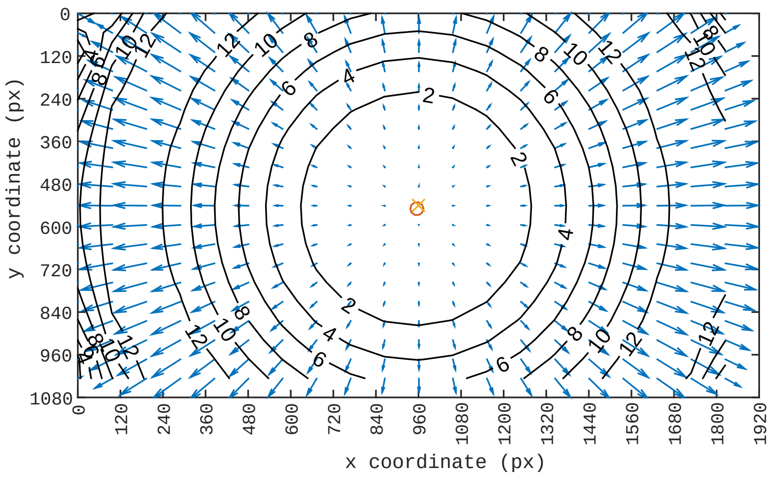
(e) : $M_S = 100$ - MPL Gyroscope



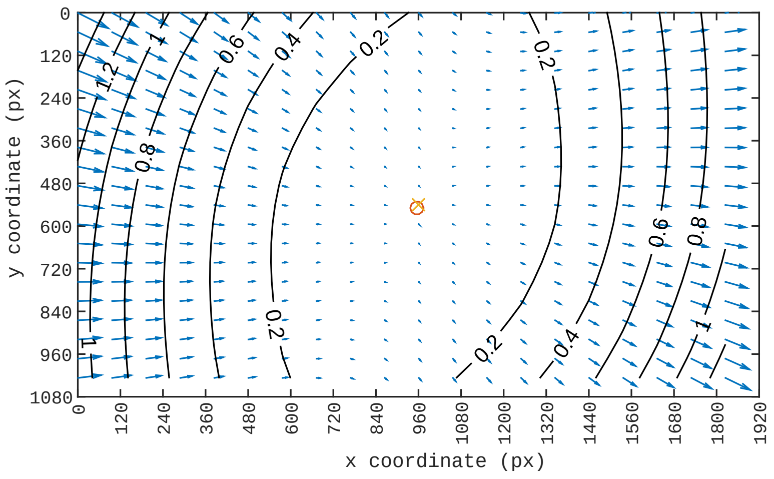
(f) : $M_S = 100$ - Corrected Gyroscope Sensor

Figure C.3: LG z-axis rotation test with filter

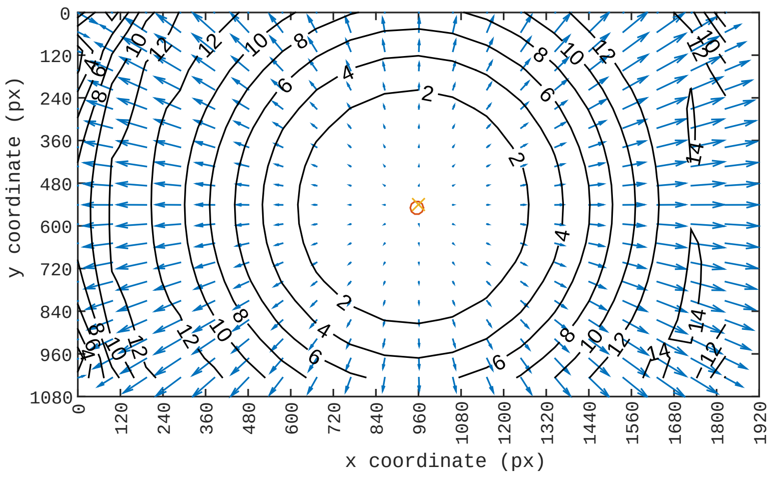
Appendix D
Samsung Galaxy S5 Back-facing Camera Distortion Model



(a) : Radial component of distortion model



(b) : Tangential component of distortion model

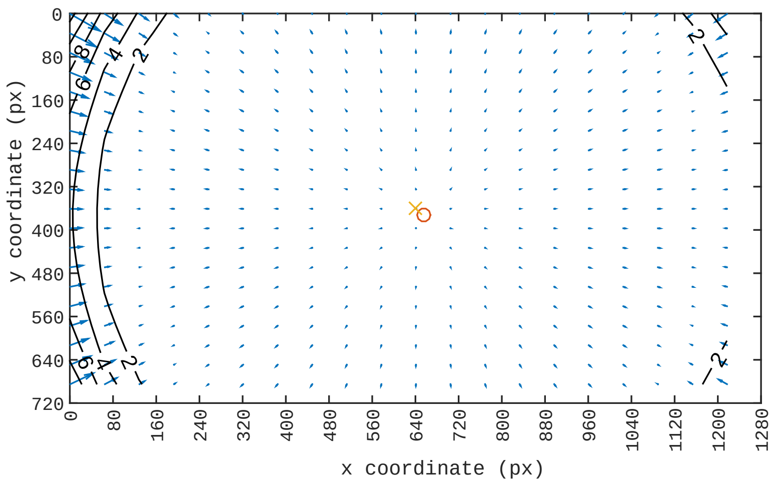


(c) : Complete distortion model

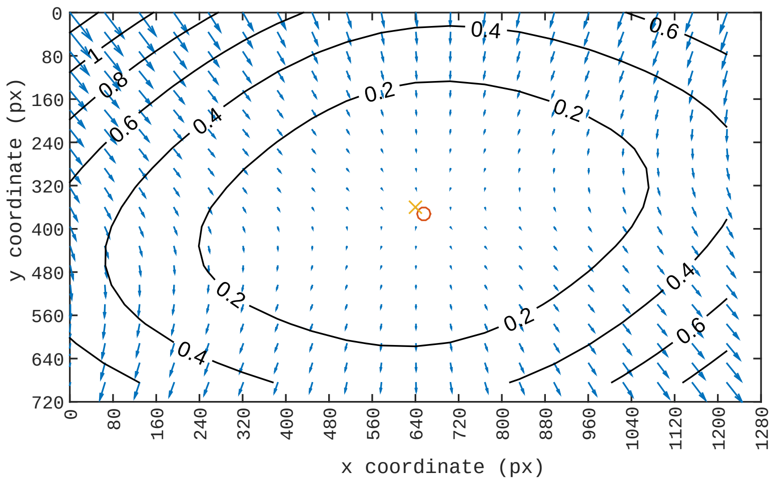
Figure D.1: Samsung Galaxy S5 real camera distortion model of back-facing camera

Appendix E

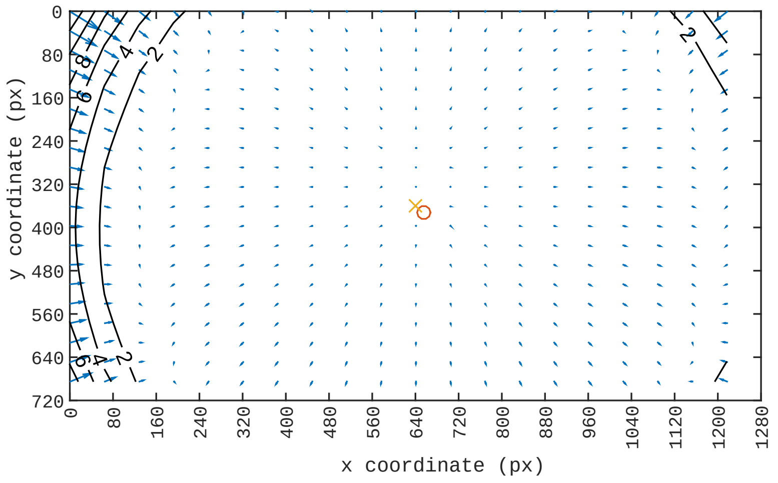
LG Optimus 2X Back-facing Camera Distortion Model



(a) : Radial component of distortion model



(b) : Tangential component of distortion model



(c) : Complete distortion model

Figure E.1: LG Optimus 2X real camera distortion model of back-facing camera

Appendix F
Content of enclosed CD-ROM

- Valasek_Jiri_BT.pdf - Electronic version of Bachelor thesis
- Naviin - Android Studio project
- gyroToVideo.m - Matlab® function.
- phoneOnMars.m - Matlab® m-script.
- sensorDataPlotter.m - Matlab® function.
- Samsung Galaxy S5 Augmented video - folder
    - Calib_Results_Samsung.mat - Camera calibration results.
    - Description_Samsung.txt - Description of captured sensors.
    - Logfile_Samsung.txt - Output Sensor data logs CSV file.
    - RecordingStart_Samsung.txt - Video recording start timestamp.
    - VideoAugmented_Samsung.mp4 - Augmented video.
    - VideoOutput_Samsung.mp4 - Recorded video.
- LG Optimus 2X Augmented video - folder
    - Calib_Results_LG.mat - Camera calibration results.
    - Description_LG.txt - Description of captured sensors.
    - Logfile_LG.txt - Output Sensor data logs CSV file.
    - RecordingStart_LG.txt - Video recording start timestamp.
    - VideoAugmented_LG.mp4 - Augmented video.
    - VideoOutput_LG.mp4 - Recorded video.