Navigation using Inertial Sensors

S. Kronemeijer

TUDelft

Bachelor Thesis

by

S. Kronemeijer

to obtain the degree of Bachelor of Science
at the Delft University of Technology,

Student number:	4709349	
Project duration:	April 19, 2022 – July 7, 2022	
Supervisor:	Dr. ir. A.R.P.J. Vijn,	TU Delft
Committee:	Dr. ir. A.R.P.J. Vijn,	TU Delft
	Dr. ir. M.B. van Gijzen,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

The world's reliability on the Global Position System (GPS) is experiencing more and more vulnerabilities. Not just environmental factors are responsible for GPS inaccuracies, but blocking or spoofing location signals has become more commonly available worldwide. This raises interest in other methods to navigate without 3rd party connections. Using an Inertial Navigation System (INS) is such a method. In this paper an algorithm is created based on the sensor fusion input of acceleration and orientation data. In order to correct itself from integration drift the algorithm makes use of Kalman filtering. Multiple simulations and real-world experiments have been done with the use of a tablet. This has promising results in the computer simulations but showcases some real difficulties when used in practice. This report gives a basis of research in this field, and many more recommendations of extensions are made.

Acknowledgements

This thesis was written as part of obtaining the bachelor's degree *Technische Wiskunde*(Applied Mathematics) at the Delft University of Technology. This project would not have been possible without my supervisor Aad Vijn, who despite his busy schedule always found the time to meet every week to rethink the formula's and computational implementation of these. These hours were the most productive and after leaving from the meeting room I would stand in the elevator with more excitement than before. This project has re-ignited my enthusiasm for mathematics and technical education, and this could not have been without Aad. During this project, I have also learned about my own way of working, slowly growing the discipline and ability to choose for studying instead of partying. After 4 years, you would expect one to know that studying at home is never as effective as opposed to going to the campus. Even though the resulting conclusion of the report is not as impressive as we might have hoped, this would also be a lesson in university research. Lastly, I would like to thank all the students, my roommates, friends and family for either studying together or keeping up with my enthusiastic ramblings every week.

S. Kronemeijer
Delft, July 2022

Contents

1 Introduction

The Global Positioning System (GPS) provides countless people all over the world the ability to determine their location using satellite connections, with an accuracy of up to 30 cm [4]. However, this United States government owned system is far from perfect. The accuracy of GPS can differ depending on multiple factors, including satellite geometry, signal blockage, atmospheric conditions, and receiver design features/quality [4]. Not just environmental factors can influence the GPS' ability, the act of 'spoofing' GPS data, where a third party generates a fake GPS signal to override the actual location, is becoming more frequent and cheaper to do. Nowadays hackers can obtain the necessary hardware for under $ 300 [14].

Because of the reliance on good conditions and connectivity, the usage of GPS for establishing ones location can be viewed as vulnerable and unreliable, for example in big warehouses, underground or in military operations. For this reason there is an increasing interest in research towards alternative ways to establish or check ones exact location. One of the most accessible ways is in the form of using an Inertial Navigation System, which will be abbreviated as INS in this report. An INS uses both accelerometers and gyroscopes to provide an angle and velocity of the device, which consequently can be used to calculate location data. Present-day, there is an INS in a lot of the devices we see around ourselves. Phones, Wii-controllers and Virtual Reality hardware are all using an INS in order to operate. This makes learning and working with INS hardware very accessible and easy to do.

In this report, a mathematical explanation will be given to the workings of such an INS using the data from the INS in a tablet. The scope of this report will be limited to the attempt of creating a model working together GPS in order to calculate the location data of this tablet as accurately as possible. Because of the imperfectness of the sensors and the software used (Matlab), a mathematical way of filtering data, called a Kalman Filter will be explored and tested in different scenario's.

2 Characterization of the Sensors

In order to understand the true mechanics that happen in an Inertial Navigation System (INS), a characterization and explanation of the sensors is given first. In this chapter, the advantages and disadvantages of the used sensors are given. After reading this chapter the user should have more in-depth knowledge of the complications the hardware provides

2.1. Accelerometer

An Accelerometer is a piece of hardware capable of measuring acceleration. Although this might seem trivial, and some might have already worked with one of these, most people do not know how an accelerometer actually works. Accelerometers are used in various places, from medical usage in modern CPR to the prediction of earthquakes and volcanic eruptions [16]. Still, the most known usage of accelerometers is probably in all modern smartphones, tablets and even laptops. Here, the accelerometer is used to determine which way is up, making it possible for users to watch videos or read text while holding the phone sideways.

The basic principle of accelerometers is made up of 2 parts: a housing that is attached to the device and does not move and a spring attached to a mass that moves freely. The force on the mass can be measured by measuring the length of the spring. If 3 of these accelerometers are combined, acceleration forces in all x, y, and z directions can be measured. If the device is lying down at rest, the accelerometer will only measure a force of gravity $g(\approx 9.81)$ downwards in the z-direction. This means that if an accelerometer were to be in free fall, all measurements would be zero.

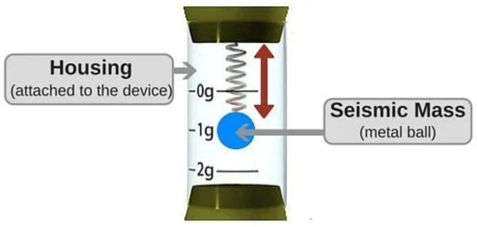


Figure 2.1: Schematic illustration of an accelerometer. This measures the acceleration in one direction by using the stretch of the spring attached (source: [6])

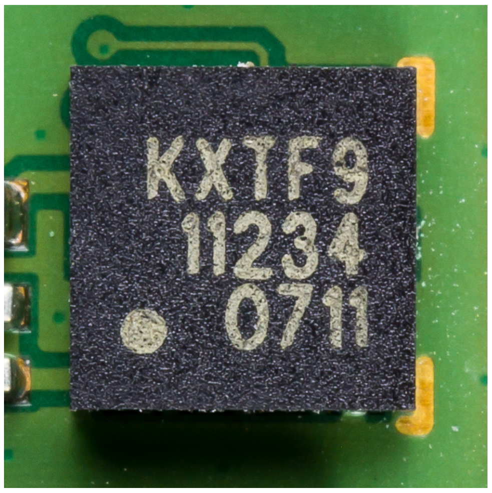


Figure 2.2: Example of an accelerometer chip in a phone (Motorola Xoom Kionix KXTF9) (source: [3])

Scientists have been improving accelerometers over the past decennia, thus present-day sensors are more complicated than the description given above. Accelerometers in phones are small chips with a size of under 10 by 10 millimeters. If the reader is interested in the different types, the electronic workings, and the impressive ways of assembly of accelerometers, there are some great sources on the internet, referenced in the bibliography[6] [16]. For now, this is outside the scope of this report.

For the purpose of navigation, the accelerometer can thus provide an acceleration proportional to the device. This is when the realisation comes that the acceleration of the device is not given a direction until the orientation of the phone is determined. For this purpose, we use another sensor in the device: the Gyroscope.

2.2. Gyroscope

The first variant of a gyroscope was used in 1743 by the English captain John Serrons to find the Horizon at sea [11]. Since then, there have been various different forms of gyroscopes used for research and experimentation, all of them relying on the same principle. A gyroscope is a device that uses the conservation of angular momentum as described in Newton's third law of motion, to create a rotor whose orientation is not affected when the rest of the device rotates. With this rotor as a reference point, it is possible to determine the angular velocity of the device, in other words, the speed at which the rotation in all 3 directions occurs.

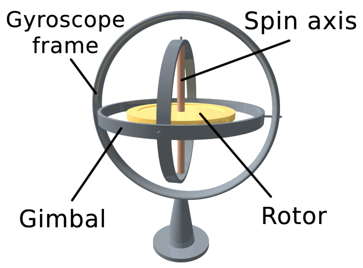


Figure 2.3: By Lucas Vieira - Own work, Public Domain (source: [2])

Just like the accelerometer, the gyroscope has applications in many different fields: Image stabilization of a camera and altitude determination in an aircraft are just some examples. Most phones nowadays also have a gyroscope in the form of a chip embedded in their hardware, this is used in mobile games or 360 degree videos or photo's, for example.
The angular velocity data is subsequently processed by means of integration in order to generate an orientation vector in the device. This can then be read out by the user. For the measurements in the coming chapters, this orientation vector is used, as its calculation is already done by the built-in software of the tablet. The gyroscope should be calibrated to return $\begin{bmatrix}0\\0\\0\end{bmatrix}$ when it is completely level with the ground and the top of the device is pointed towards the North.

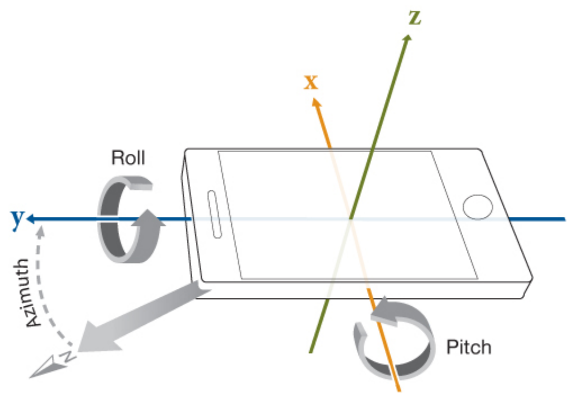


Figure 2.4: The directions of the accelerometer (x,y,z) and the gyroscope(Azimuth, Pitch, Roll) visualised (source: [9]

2.3. Characterization

The INS used here provides the user with 6 different data points: 3 from the accelerometer and 3 from the orientation. These are given as X, Y, Z entries in a timetable as shown in the Figure 2.7 below. The accelerometer gives data using m/s^2 as units, where the orientation is given as coordinates between -180 and 180 degrees for the azimuth, pitch, and roll respectively.

In order to determine the accuracy of these sensors, an experiment is performed where the INS is placed on a flat surface for 2 minutes keeping any disturbance to the INS or the surface to a minimum. In a perfect world, this would result in 2 minutes of exactly the same value for all 6 degrees of freedom, yet the results show otherwise. In Figure 2.5 the results of the experiment is shown in the form of a histogram. The data is shown to be quite Gaussian using a fit. The x and y data is expected to be zero, where the acceleration in the z-direction is expected to have a mean of 9.81. However, this is not happening here. The reason this is not the case could be because of the surface not being completely flat, which could cause the gravitational vector to be measured in other directions than z. Also, the sensors could possibly have a bias resulting in a standard error. Most plausible is that both reasons play a part here.

The gyroscope sensor gives data helping to explain the inaccuracy of the acceleration data. As shown in Figure 2.6 the orientation of the INS is not even close to having zero for the pitch and roll averages. This supports the reasoning before, which was suggesting that the surface is not flat. Although this is an angle of just under 1 degree in the y-direction, the relative size of the gravitational force could possibly still have an impact. Secondly, we see that the orientation in the x-direction, the azimuth, looks like it is very stable compared to the other directions. This is untrue though, as the histogram function creates bigger bins here as the difference between the highest and lowest value is of a much higher magnitude than the y- and z-direction.

In conclusion, the sensors for acceleration and orientation are experiencing a lot of noise and are presumably having a bias, as these consumer-market sensors are not perfectly calibrated. Also, there is a clear impact of the orientation data on the acceleration data resulting from the gravitational force.

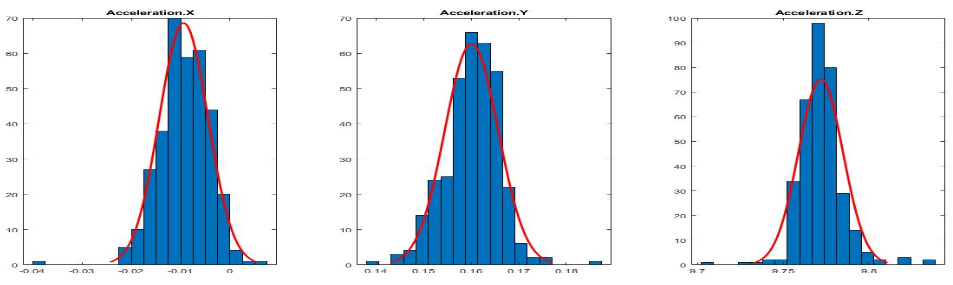


Figure 2.5: Histogram of the accelerometer data for 2 minutes of a stationary situation and a Gaussian fit to the data.

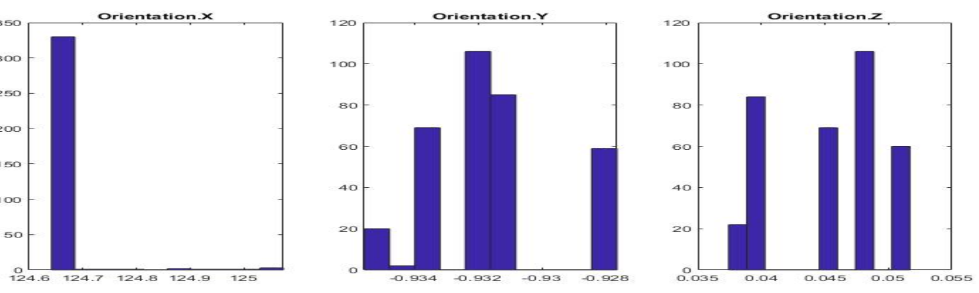
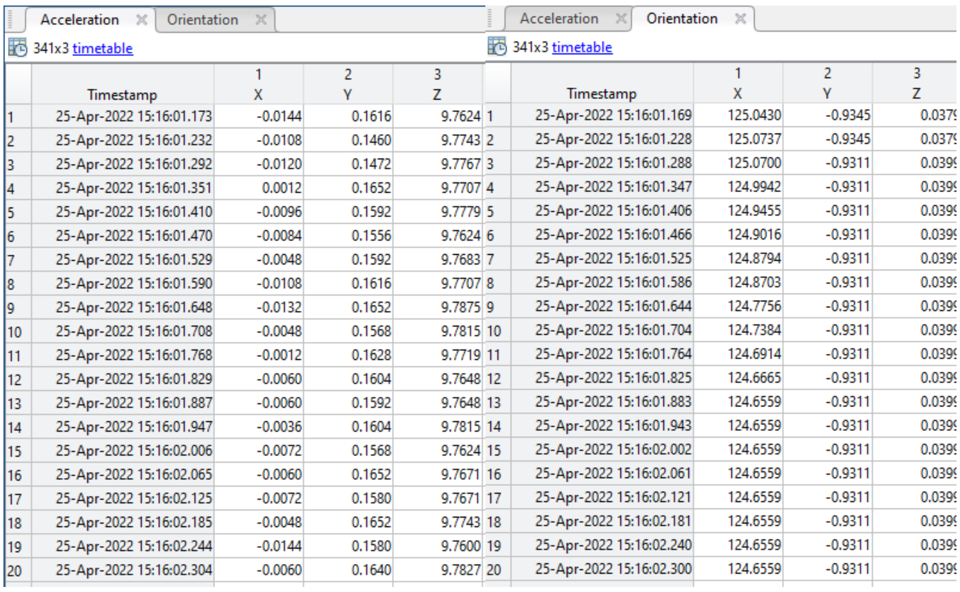


Figure 2.6: Histogram of the orientation data for 2 minutes of a stationary situation.

In this test some important values have been assessed. The mean of the accelerometer data is determined while the INS was not moving. This indicates some sort of bias. Besides the mean, the variance of the Gaussian fit is useful for simulating similar accelerometer data in Chapter 3. The results are shown in a table.

	X	Y	Z
Mean	-0.0094	0.1601	9.7719
Variance	0.0050	0.0056	0.0130


Acceleration | Orientation

341x3 timetable

	Timestamp	1 X	2 Y	3 Z
1	25-Apr-2022 15:16:01.173	-0.0144	0.1616	9.7624
2	25-Apr-2022 15:16:01.232	-0.0108	0.1460	9.7743
3	25-Apr-2022 15:16:01.292	-0.0120	0.1472	9.7767
4	25-Apr-2022 15:16:01.351	0.0012	0.1652	9.7707
5	25-Apr-2022 15:16:01.410	-0.0096	0.1592	9.7779
6	25-Apr-2022 15:16:01.470	-0.0084	0.1556	9.7624
7	25-Apr-2022 15:16:01.529	-0.0048	0.1592	9.7683
8	25-Apr-2022 15:16:01.590	-0.0108	0.1616	9.7707
9	25-Apr-2022 15:16:01.648	-0.0132	0.1652	9.7875
10	25-Apr-2022 15:16:01.708	-0.0048	0.1568	9.7815
11	25-Apr-2022 15:16:01.768	-0.0012	0.1628	9.7719
12	25-Apr-2022 15:16:01.829	-0.0060	0.1604	9.7648
13	25-Apr-2022 15:16:01.887	-0.0060	0.1592	9.7648
14	25-Apr-2022 15:16:01.947	-0.0036	0.1604	9.7815
15	25-Apr-2022 15:16:02.006	-0.0072	0.1568	9.7624
16	25-Apr-2022 15:16:02.065	-0.0060	0.1652	9.7671
17	25-Apr-2022 15:16:02.125	-0.0072	0.1580	9.7671
18	25-Apr-2022 15:16:02.185	-0.0048	0.1652	9.7743
19	25-Apr-2022 15:16:02.244	-0.0144	0.1580	9.7600
20	25-Apr-2022 15:16:02.304	-0.0060	0.1640	9.7827

Acceleration | Orientation

341x3 timetable

	Timestamp	1 X	2 Y	3 Z
1	25-Apr-2022 15:16:01.169	125.0430	-0.9345	0.0379
2	25-Apr-2022 15:16:01.228	125.0737	-0.9345	0.0379
3	25-Apr-2022 15:16:01.288	125.0700	-0.9311	0.0399
4	25-Apr-2022 15:16:01.347	124.9942	-0.9311	0.0399
5	25-Apr-2022 15:16:01.406	124.9455	-0.9311	0.0399
6	25-Apr-2022 15:16:01.466	124.9016	-0.9311	0.0399
7	25-Apr-2022 15:16:01.525	124.8794	-0.9311	0.0399
8	25-Apr-2022 15:16:01.586	124.8703	-0.9311	0.0399
9	25-Apr-2022 15:16:01.644	124.7756	-0.9311	0.0399
10	25-Apr-2022 15:16:01.704	124.7384	-0.9311	0.0399
11	25-Apr-2022 15:16:01.764	124.6914	-0.9311	0.0399
12	25-Apr-2022 15:16:01.825	124.6665	-0.9311	0.0399
13	25-Apr-2022 15:16:01.883	124.6559	-0.9311	0.0399
14	25-Apr-2022 15:16:01.943	124.6559	-0.9311	0.0399
15	25-Apr-2022 15:16:02.002	124.6559	-0.9311	0.0399
16	25-Apr-2022 15:16:02.061	124.6559	-0.9311	0.0399
17	25-Apr-2022 15:16:02.121	124.6559	-0.9311	0.0399
18	25-Apr-2022 15:16:02.181	124.6559	-0.9311	0.0399
19	25-Apr-2022 15:16:02.240	124.6559	-0.9311	0.0399
20	25-Apr-2022 15:16:02.300	124.6559	-0.9311	0.0399

Figure 2.7: The raw data obtained from the accelerometer and gyroscope, note how the accelerometer and gyroscope provides data points every 0.02 seconds

3 The mathematical model

In order to create a model to approximate the location of a moving INS over time, there are some necessary steps to be taken before we have our expression. In this chapter we will first create a rotational matrix to convert our reference frame of the device to a reference frame of a map, like google earth. Afterwards we are able to remove the gravitational force and the bias established in the previous chapter. After all this is done, we are able to use a technique called *dead reckoning* to finally come to an expression of a linear system.

3.1. The rotational matrix

Because the acceleration vector obtained by the accelerometer is proportional relative to the orientation of the device, it is said to exists in the *reference frame* of the device. We give this vector therefore the superscript of d for device. The actual vector, as seen in the reference frame of the earth is given the superscript of e. Thus we have two vectors $\vec{x}^d$ and $\vec{x}^e$. In order to gain $\vec{x}^e$ from the device vector, a transformation is made using a rotational matrix $\boldsymbol{R}^{ed}$. Thus, $\vec{x}^e = \boldsymbol{R}^{ed}\vec{x}^d$. This rotational matrix will be determined.

The goal of this rotational matrix is to use the values for the azimuth(ψ), pitch(θ), and roll(ϕ) as described in the previous chapter to rotate to the x, y, and z plane of the earth. But before rotation is in order, first this earth reference frame has to be established. Because of the small scope of this research, the curve of the earth is not included. Also the assumption is made that the world is completely flat. This leads us to the reference frame where the z-axis is straight up from the earth. We choose the y-axis to be pointing towards the north and the x-axis eastwards. This implies that $\vec{x}^e = \vec{x}^d$ if all of the orientation angles are zero.

Since the orientation angles are usually not all zero, the rotational matrix is necessary. Changing the orientation of a vector to another frame is equal to the change of the reference frame. Hence, we can express the rotational matrix in 3 rotations. These are the 3 rotations around the 3 axis as shown in Figure (3.1). These rotations yield the rotation matrices shown below. The derivation of $\boldsymbol{R}_\psi$ is given as example.

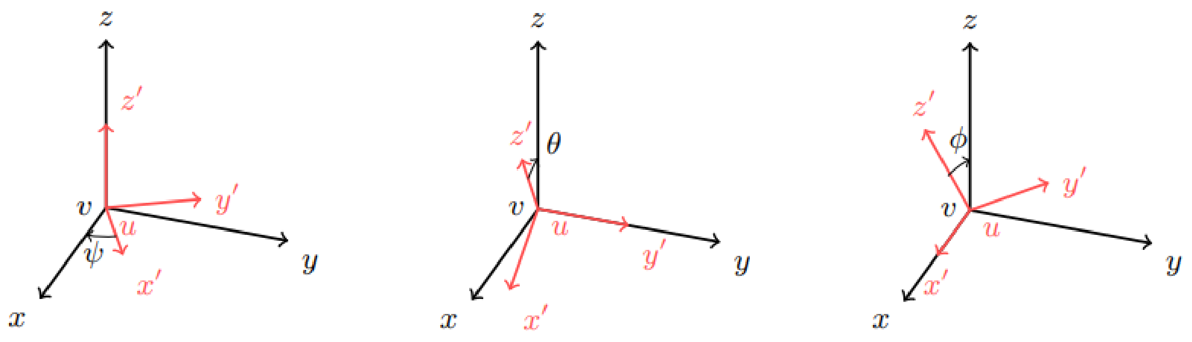


Figure 3.1: The rotations of the 3 different angles, θ, ψ, ϕ

As seen in Figure (3.1), the new x-axis x' is obtained by rotating an angle of ψ. This means that the location of x' in this 3-dimensional frame equal is to $[\cos\psi \; \sin\psi \; 0]^T$ using the frame of x, y and z as shown in black. In a similar fashion, y' is also an angle of ψ from y, thus basic trigonometrics gives $[-\sin\psi \; \cos\psi \; 0]^T$. Finally, as $z' = z$, this leads to a simple vector of $[0 \; 0 \; 1]^T$. When we concatenate these 3 matrices it provides us with the first rotational matrix $\boldsymbol{R}_\psi$.

$$\boldsymbol{R}_\psi = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{R}_\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$\boldsymbol{R}_\phi = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}$$

Multiplying these 3 rotations gives the desired $\boldsymbol{R}^{ed}$, which will just be called $\boldsymbol{R}$ from now on.

$$\begin{aligned}
\boldsymbol{R} &= \boldsymbol{R}_\psi \boldsymbol{R}_\theta \boldsymbol{R}_\phi \\
&= \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \\
&= \begin{bmatrix} \cos\phi\cos\psi - \sin\phi\cos\theta\sin\psi & \cos\phi\sin\psi + \sin\phi\cos\theta\cos\psi & \sin\phi\sin\theta \\ -\sin\phi\cos\psi - \cos\phi\cos\theta\sin\psi & -\sin\phi\sin\psi + \cos\phi\cos\theta\cos\psi & \cos\phi\sin\theta \\ \sin\theta\sin\psi & -\sin\theta\cos\psi & \cos\theta \end{bmatrix}
\end{aligned}$$

3.2. Removal of gravity vector and bias

There is an argument to be made about removing the bias before changing the reference frame. The bias is formed in the initial measurements after all, in the device reference frame. Nevertheless, the gravity vector has to be compensated after the rotation to the earth frame, it is therefore easier to do both afterwards instead of separately. The biases form a vector in the body frame $[e_x^b \; e_y^b \; e_z^b]^T$, this is including the gravity vector g. Using the rotation matrix $\boldsymbol{R}$ we get $\vec{e} = [e_x^e \; e_y^e \; e_z^e]^T = \boldsymbol{R}[e_x^b \; e_y^b \; e_z^b]^T$. Removing the left-hand side of the equation from $\vec{x}^e$ gives us the method used in this paper to attempt to remove the initial bias.

3.3. Dead-reckoning

So far, we have used the orientation data from the gyroscope to bring the acceleration vector into the earth reference frame using the rotational matrix. Then we removed the biases and now we are ready to calculate positions. The approach used here is called *dead reckoning.*

Dead reckoning assumes it knows the previous location, and then uses a vector containing direction and speed to calculate an estimate for the next location. In the case of navigation with an INS the next location can be determined using Newton's equation of motion:

$$P_n = P_{n-1} + tV_{n-1} + \frac{t^2}{2}A_{n-1} \tag{3.1}$$

Where P_n is the position at a point n in time, P_{n-1} is the previous point, t is the time in seconds between two measurements n and $n-1$, V is the speed in m/s at the previous measurement as a vector, and A_{n-1} the acceleration in m/s^2 at the previous measurement as a vector. Since V_{n-1} is still unknown we use another form of linear physics to calculate this in a similar fashion:

$$V_n = V_{n-1} + tA_{n-1} \tag{3.2}$$

This presents a system of equations where the location P at any point is only dependent on acceleration A, provided an initial position P_0 and speed V_0 are given. It is necessary to work with V and A in the device reference frame since there could be a situation where these factors do not change but the orientation does. Hence we need to update using the rotation matrix every iteration. Let V and A be in the device reference frame. Since P is in the earth reference frame, Equation (3.1) is adjusted to

$$P_n = P_{n-1} + t\boldsymbol{R}V_{n-1} + \frac{t^2}{2}\boldsymbol{R}A_{n-1} \tag{3.3}$$

Apart from using the rotation matrix $\boldsymbol{R}$, the error vector $\vec{e}$ is also to be removed from A in the equation. Remember that we use the error vector from the earth reference frame, and therefore it does not need to be rotated here.

$$P_n = P_{n-1} + t\boldsymbol{R}V_{n-1} + \frac{t^2}{2}(\boldsymbol{R}A_{n-1} - \vec{e}) \tag{3.4}$$

For Equation (3.2) this is more complicated. As V is to be in the device reference frame, but A^d is added while $\vec{e}$ is to be subtracted from A^e. In order to solve this problem the inverse of $\boldsymbol{R}$ is used as shown below.

$$V_n = V_{n-1} + t\boldsymbol{R}^{-1}(RA_{n-1} - \vec{e}) \tag{3.5}$$

Equations (3.4) and (3.5) are the basis of the mathematical model.

3.4. system expression

We want to transform te systems of equations above towards an expression that is more friendly to work with and easier to compute. Therefore we are looking for a discrete, state space model. This looks like this:

$$\begin{cases} \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k \end{cases}$$

Here $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are matrices, $\mathbf{x}(k)$ is the state at time k, $y(k)$ is the output of the system at time k, and $\mathbf{u}(k)$ is the input of the system at time k. The system is called discrete because the value of k is an integer and the difference between $\mathbf{x}(k)$ and $\mathbf{x}(k+1)$ is not continuous.

For the case of our navigation problem the following state is chosen:

$$\vec{\mathbf{x}}_k = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \end{bmatrix}$$

Note that the dot on top of the variable indicates the derivative of the location, i.e. the speed V.
The input of the system $\mathbf{u}(k)$ is the acceleration, hence

$$\mathbf{u}_k = \begin{bmatrix} \ddot{x}_k \\ \ddot{y}_k \\ \ddot{z}_k \end{bmatrix}$$

If we return to Equations 3.1 and 3.2, we see that we can write this as a set of equations.

$$\begin{cases} x_k = x_{k-1} + t\dot{x}_{k-1} + \frac{t^2}{2}\ddot{x}_{k-1} \\ y_k = y_{k-1} + t\dot{y}_{k-1} + \frac{t^2}{2}\ddot{y}_{k-1} \\ z_k = z_{k-1} + t\dot{z}_{k-1} + \frac{t^2}{2}\ddot{z}_{k-1} \\ \dot{x}_k = \dot{x}_{k-1} + t\ddot{x}_{k-1} \\ \dot{y}_k = \dot{y}_{k-1} + t\ddot{y}_{k-1} \\ \dot{z}_k = \dot{z}_{k-1} + t\ddot{z}_{k-1} \end{cases}$$

or

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k$$

$$\begin{bmatrix} x_k \\ y_k \\ z_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t & 0 & 0 \\ 0 & 1 & 0 & 0 & t & 0 \\ 0 & 0 & 1 & 0 & 0 & t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ z_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \\ \dot{z}_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{t^2}{2} & 0 & 0 \\ 0 & \frac{t^2}{2} & 0 \\ 0 & 0 & \frac{t^2}{2} \\ t & 0 & 0 \\ 0 & t & 0 \\ 0 & 0 & t \end{bmatrix} \begin{bmatrix} \ddot{x}_n \\ \ddot{y}_n \\ \ddot{z}_n \end{bmatrix}$$

The second equation is going to provide the output of the system. The output of the system should be equal to the position of the INS and is therefore easily available as the information is already included in the state. This makes the factor with matrix D obsolete and this can thus be removed.

$$\begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ z_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \end{bmatrix}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k$$

This gives us the values of A, B, C in the simplest model, however the rotation still has to be included. We do this as follows:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & & & \\ 0 & 1 & 0 & & [tR] & \\ 0 & 0 & 1 & & & \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} & \left[\frac{t^2}{2}R\right] & \\ & & \\ \mathrm{t} & 0 & 0 \\ 0 & \mathrm{t} & 0 \\ 0 & 0 & \mathrm{t} \end{bmatrix}$$

where R is the rotational matrix as described in Section (3.1). To remove the bias, we simply subtract it in the end. Let us call this vector $\mathbf{W}$.

$$\mathbf{W} = \begin{bmatrix} \left[\frac{t^2}{2}(R^{-1}\vec{e})\right] \\ \left[t(R^{-1}\vec{e})\right] \end{bmatrix}$$

This leaves us with this final system of estimation:

$$\begin{bmatrix} x_k \\ y_k \\ z_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & & & \\ 0 & 1 & 0 & & [tR] & \\ 0 & 0 & 1 & & & \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ z_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \\ \dot{z}_{k-1} \end{bmatrix} + \begin{bmatrix} & & \\ & \left[\frac{t^2}{2}R\right] & \\ & & \\ t & 0 & 0 \\ 0 & t & 0 \\ 0 & 0 & t \end{bmatrix} - \begin{bmatrix} \left[\frac{t^2}{2}(R^{-1}\vec{e})\right] \\ \left[t(R^{-1}\vec{e})\right] \end{bmatrix}$$

$$\begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ z_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \end{bmatrix}$$

or

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k - \boldsymbol{W}$$
$$\boldsymbol{y}_{k+1} = \boldsymbol{C}\boldsymbol{u}_k$$

This gives us our desired system, which we will use to simulate and experiment in the coming chapters.

4 The Kalman filter

During this chapter, a brief summary will be given on the general idea of the Kalman filter. This is a filtering algorithm named after Rudolf E. Kálmán, a Hungarian-American mathematician who was one of the primary developers of its theory. [8].

4.1. The problem

There are situations where you try to determine an output from a dynamic model, such as described in the previous chapter. If the input of a system and the model you devised are both perfectly reliable, this would lead to the true value of the state at any point in time. Unfortunately the world is not so kind. The input of a system can be different from the actual input because of uncertainty of the sensors due to calibration or rounding errors. This error in the input is called *Estimation noise.* Besides from the estimation noise, there is also the *Process noise*: The dynamic model is not always perfect, it usually contains some estimation, something that was not taken into account for dealing with, or again, rounding errors on the computational side. Now assume that there is another way of measuring the output the model is trying to approximate. This so called *sensor-fusion* might give help to reduce the error. This measurement might give some different output than the dynamic model does, but these measurements also have their own *Measurement noise.* Which of the methods do you trust? The estimation from the dynamic model? The measurements ? Or do you take the average? The Kalman filter is one of the most used algorithms to predict the next state taking both methods and their uncertainty all into account.

4.2. How it works

The Kalman filter works on the basis of some prerequisites. First of all, there must be a dynamic system in place that does the estimation and an initial state must be given. It is necessary that this system is of the form of an system as considered in Section 3.4. Furthermore, there has to be some sort of measurement data in place which the result can be compared to. Lastly, the noise has to either be known, or an educated guess can be made. This is the case for both the measurement, estimate and process uncertainty.
Once these requirements are met, the algorithm can be used. The algorithm consists of 2 steps: the prediction step and the update step. Both will be explained.

During the prediction step a new estimate is made. This is done in the first calculation of the dynamic model.

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k$$

During the prediction step the estimate uncertainty also gets updated.

$$\boldsymbol{P}_{k+1} = \boldsymbol{A}\boldsymbol{P}_k\boldsymbol{A}^T + \boldsymbol{Q}$$

This equation requires some explanation. $\boldsymbol{P}_k$ shows the estimate uncertainty. This uncertainty is expressed as the covariance matrix, in the three-dimensional case it looks like this.

$$\boldsymbol{P} = COV(x, y, z) = COV(\boldsymbol{x}) = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yz}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 & \sigma_{zz}^2 \end{bmatrix}$$

Here σ_{xx}^2 is the variance of the variable x, or the standard deviation σ_{xx} squared. The variables with 2 different subscripts, like σ_{xy}^2 show the covariance between x and y, two different values in the vector $\boldsymbol{x}$. For the derivation of $\boldsymbol{P}_{k+1}$ we use some arithmetic from the field of probability.

$$\begin{aligned}
\boldsymbol{P}_{k+1} &= COV(\boldsymbol{x}_{k+1}) \\
&= E((\boldsymbol{x}_{k+1} - \boldsymbol{\mu}_{k-1})(\boldsymbol{x}_{k+1} - \boldsymbol{\mu}_{k-1})^T) \\
&= E((\boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k - \boldsymbol{A}\boldsymbol{\mu}_k - \boldsymbol{B}\boldsymbol{u}_k)(\boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k - \boldsymbol{A}\boldsymbol{\mu}_k - \boldsymbol{B}\boldsymbol{u}_k)^T) \\
&= E((\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{A}\boldsymbol{\mu}_k)(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{A}\boldsymbol{\mu}_k)^T) \\
&= E(\boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{\mu}_k)(\boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{\mu}_k))^T) \\
&= E(\boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{\mu}_k)(\boldsymbol{x}_k - \boldsymbol{\mu}_k)^T \boldsymbol{A}^T) \\
&= \boldsymbol{A}E((\boldsymbol{x}_k - \boldsymbol{\mu}_k)(\boldsymbol{x}_k - \boldsymbol{\mu}_k)^T)\boldsymbol{A}^T \\
&= \boldsymbol{A}\boldsymbol{P}_k\boldsymbol{A}^T
\end{aligned}$$

We then add the process uncertainty Q to P_{k+1} for this is also part of the uncertainty of the dynamic system its output.

The update equation is where our measurements are included in the algorithm. First we compute the *Kalman gain* $\boldsymbol{K}$ which can be seen as the ratio between the uncertainty of the estimate and the uncertainty of the measurement. $\boldsymbol{K}$ is a matrix with values between zero and 1. If $\boldsymbol{K}$ is close to 1, we rely almost completely on the estimation, whereas a $\boldsymbol{K}$ close to 0 indicates that we trust the measurements much more.

$$\boldsymbol{K} = \boldsymbol{P}_k\boldsymbol{C}^T(\boldsymbol{C}\boldsymbol{P}_k\boldsymbol{C}^T + \boldsymbol{R})^{-1}$$

Here we use the observation matrix $\boldsymbol{C}$ to extract the observable information from the measurement uncertainty $\boldsymbol{R}$. Afterwards we use the Kalman gain to calculate a new state $\boldsymbol{x}_{k,new}$.

$$\boldsymbol{x}_{k,new} = \boldsymbol{x}_{k,old} + \boldsymbol{K}(\boldsymbol{z}_k - \boldsymbol{C}\boldsymbol{x}_{k,old}) \tag{4.1}$$

Where $\boldsymbol{x}_{k,old}$ is the first estimate that was calculated in the dynamic model during the prediction step. $\boldsymbol{z}_k$ is the measurement data at moment k. Here we confirm that in the case of a higher $\boldsymbol{K}$, the measurement is more influential than the estimation and otherwise for lower $\boldsymbol{K}$
The last equation of the update step is where we update the estimate uncertainty $\boldsymbol{P}_k$.

$$\boldsymbol{P}_{k,new} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\boldsymbol{P}_{k,old}(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})^T + \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^T$$

4.3. Application in this system

In the case of our navigation problem, we need to make some small adjustment to the before-mentioned system. First of all, as we have seen in chapter 3, we are subtracting the bias during the estimation step.

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k - \boldsymbol{w}$$

This modification does not affect the rest of the workings of the Kalman filter. In addition, the process uncertainty $\boldsymbol{Q}$, the measurement uncertainty $\boldsymbol{R}$ and the initial estimate uncertainty $\boldsymbol{P}_0$ are to be determined. For the process uncertainty $\boldsymbol{Q}$ we create a 6 by 6 matrix to express the confidence we have in the system and the formulas used. The values in this matrix describe the amount of distance and speed we expect to be away from the true value. In other words, if we were to have the perfect input of the dynamic model, how accurate is the output? Since we know that Newton's equations of motion rely on the assumption of constant acceleration, there is some uncertainty there. Also, in our model we do not take other things into account such as the curvature of the earth or the Coriolis-effect. This makes it challenging to choose correct values for $\boldsymbol{Q}$.

For the sake of simplicity, the assumption was made to have the parameters of state $\boldsymbol{x}$ be independent of each other. Hence, all the values of $\boldsymbol{Q}$ that are not in the diagonal are set to zero. Furthermore, the values for location and speed in all 3 directions are chosen to be equal. This is often mainly done to ease the calculation process, but can be justified when considering that the calculations for all 3 directions are similar and the noise of the location and speed are in the same order of magnitude. Still, we think that we have a quite accurate model. Henceforth the matrix $\boldsymbol{Q}$ is chosen as this.

$$\boldsymbol{Q} = \begin{bmatrix} 0.001 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.001 & 0 & 0 & 0 \\ 0 & 0 & 0 & 001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 001 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.001 \end{bmatrix}$$

For the measurement uncertainty $\boldsymbol{R}$, this represents the amount of trust we have in the measurements of the GPS. The GPS data is given in latitude and longitude coordinates, thus we let this be represented by a 2x2 matrix. Again, we let this matrix be a diagonal matrix as we assume there to be no dependencies between the variables. We suspect the GPS system to be quite reliable, but because of objects blocking the signal, the cheap and small chips used for hardware, and the rounding of latitude and longitude degrees, there is still some accuracy to be desired. Therefore we set the accuracy to 1.

$$\boldsymbol{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Lastly, the estimate uncertainty at the beginning, $\boldsymbol{P}_0$ has to be decided on. The inital estimate of the location is set at exactly the point of the first GPS data point, therefore the x and y location are set to the same accuracy as in the measurement uncertainty. The location for the height is not relevant here, so this can be left to zero. We set the initial speed to be zero, and while the attempt is done to stand completely still before starting the estimations, we can not trust this completely. This is why we let the inital horizontal and vertical speed have a variance of 0.1 m/s.

$$\boldsymbol{P}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix}$$

Once more, we assume no dependency between the uncertainties.

5 Simulated data

Now that the complete model is set up, the time has come to test the effectiveness of our model. We will first do this using simulated data, either with or without variance, and later extending it with the Kalman filter.

5.1. Why simulate data?

As we have seen in Chapter 2, the sensors used in the INS of the tablet is experiencing a bunch of noise. This makes it difficult to work with; When errors arise it is hard to tell whether the model is wrong or the sensors. For that reason we first use some simulated 'perfect' data to test the model with.

5.2. How to simulate perfect data

In our simulated data the goal is to test our methods in the different ways we could receive our information. Therefore we have 3 different tests set up: we will try to create a square with lengths 1, a unit circle (with radius 1) and a Lissajous figure.

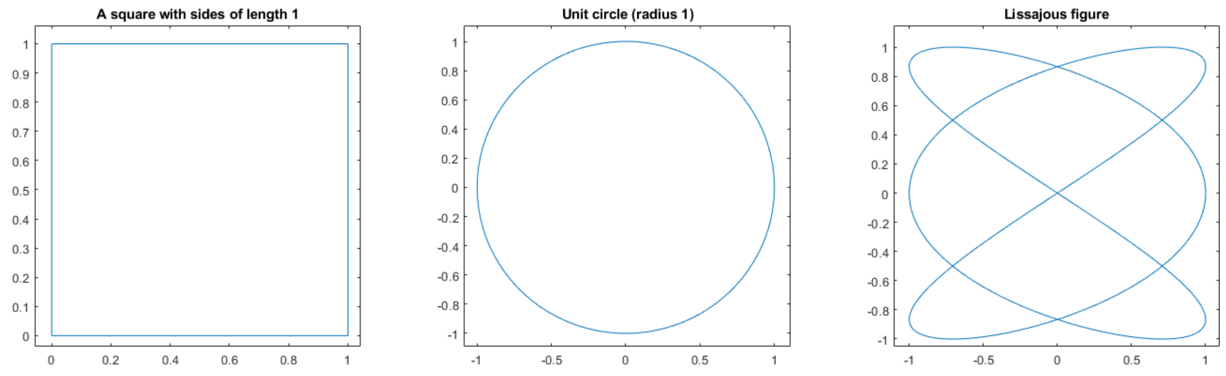


Figure 5.1: The square, circle and Lissajous figure that are to be simulated

For the square we just used linspaces, the circle is a product of parametric equations $(x(t), y(t)) = (cos(t), sin(t))$, and the Lissajous figure is defined by $(x(t), y(t)) = (cos(3t), sin(3t))$ where $0 \leq t \leq 2\pi$ for both the circle and Lissajous figure. The goal is to simulate a path where we move anti-clockwise, starting at (0,0) for the square, and (0,1) for the other two.

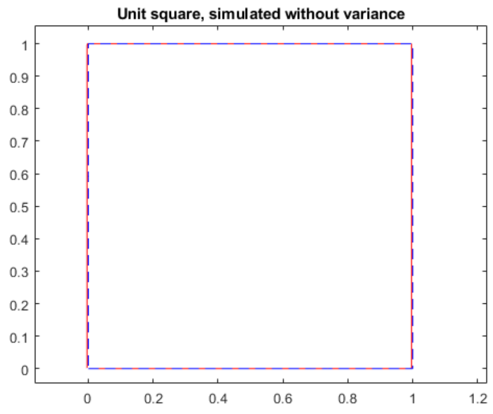
5.3. Results

We want to test our system in different circumstances thus we will be walking this path in 3 different ways, progressing to more and more complex data.

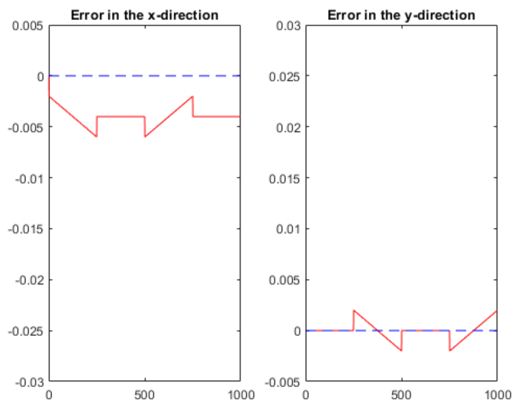
5.3.1. Walking a square

For the simplest case, we will assume that the INS is always in the same reference frame as the world reference frame. This means that the orientation vector is always equal to 0 and the rotation matrix $\boldsymbol{R}$ is equal to the identity matrix. We test our algorithm by giving it an acceleration towards (0,1) and then letting the acceleration be zero as to simulate a constant speed. Then, at the point (0,1) we decelerate to zero m/s in the x direction, while accelerating in the y direction. We repeat this 2 more times to get back to the origin.

3 tests were performed using the model described: First without any variance in the input, then extended with the variance, and lastly with the variance and the Kalman filter in place. For every test we use a thousand data points and we compare the accuracy using the Root Mean Squared Error (RMSE). The RMSE takes the error in the pythagorean of x and y to get the total error at every point, and squares this in order to obtain just positive values. Afterwards the root is taken from the sum of these total errors in order to remain the same order of magnitude as the values itself.



(a) The path visualized

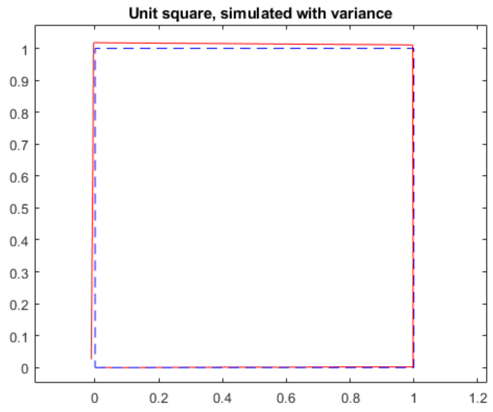


(b) The error in the x and y direction (RMSE: 0,0639)

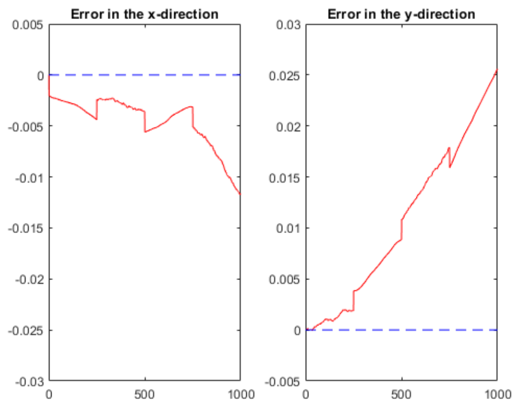
Figure 5.2: Unit square, simulated without variance

As seen in Figure (5.2) the way that the movement is simulated works very well. The error with respect to the exact unit square, as shown in blue, is small. The RMSE proofs this, as its value is only 0,0639. This shows that our model works in this case.

Below, there is Figure (5.3). The extension of adding a variance is visually clear when comparing the error margins. The error keeps growing, this could be because of *integration drift*. The RMSE here is 0,1090. This means that over a course of 4 meter, the average deviation from 'real path' is about 10 centimeters. Since we use variance values from our own INS, as described in Section 4.3, this displays the (in)accuracy of the sensors.



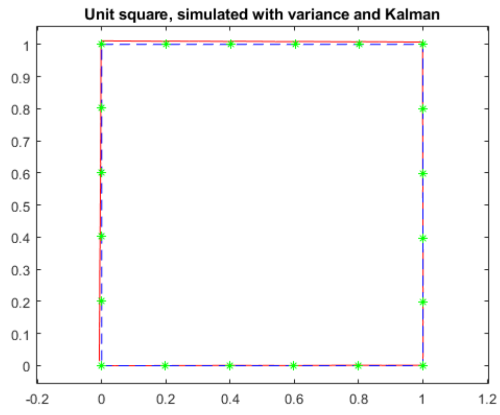
(a) The path visualized



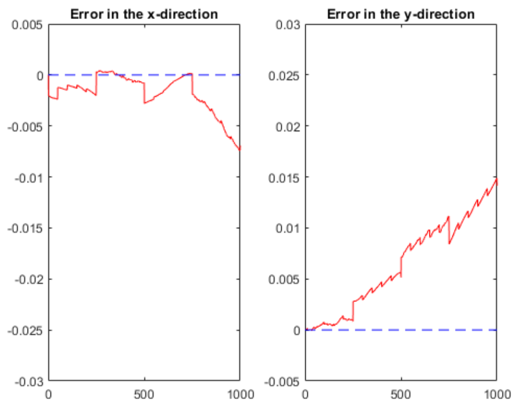
(b) The error in the x and y direction (RMSE: 0,1090)

Figure 5.3: Unit square, simulated with variance

In order to deal with the errors, the simulation is extended with a Kalman filter. For the uncertainties the values as shown in section 4.3 are used. The filter makes a small correction every 50th data point as indicated by the green dots in the graph. This mimics the experimental data, where the GPS data comes in every second and the accelerometer and gyroscope sensors every 0.02 seconds. As seen in Figure (5.4) the graph has smaller errors as before, the RMSE is 0,0833.



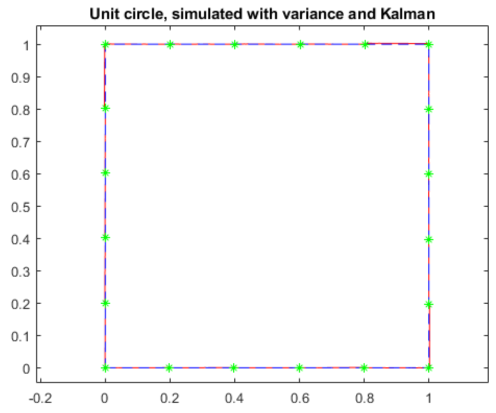
(a) The path visualized



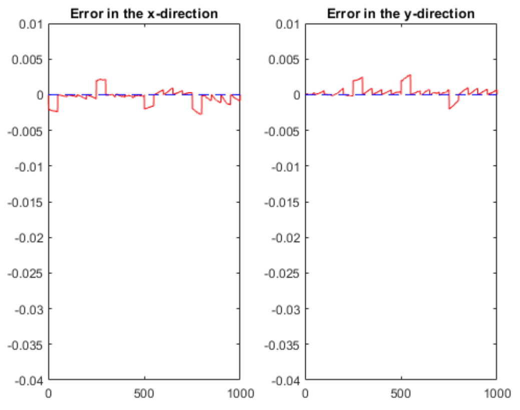
(b) The error in the x and y direction (RMSE: 0,0888

Figure 5.4: Unit square, simulated with variance and Kalman filter

Although this is an improvement, this resulting RMSE could be lower. This is because of the values chosen for the uncertainties. By use of trial-and-error we found that the RMSE is much lower (0.303) when the measurement uncertainty matrix $\boldsymbol{R}$ is set to zero. This is sensible as we take the measurement data to be the true data in this case.



(a) The path visualized

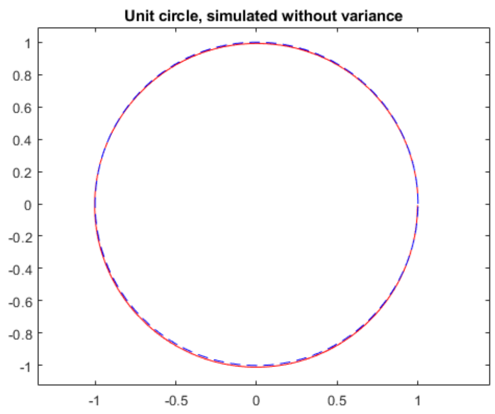


(b) The error in the x and y direction (RMSE: 0,303)

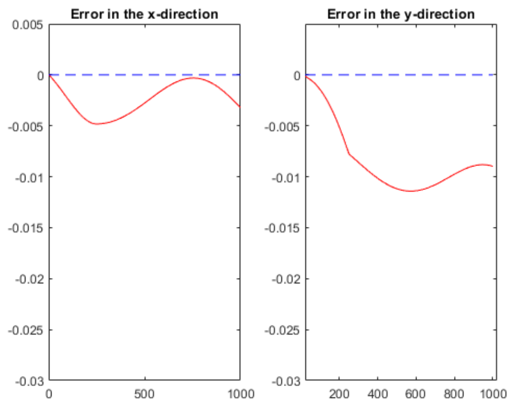
Figure 5.5: Unit square, simulated with variance and Kalman filter (Measurement uncertainty = 0)

5.3.2. The Unit circle

The path of walking along the unit circle starts at (1,0) and moves anti-clockwise. In this simulation we will be using the orientation vector instead of the acceleration to create the desired route. This means that we will start with an orientation of -90 degrees and let it decrement until -180. There we jump to 180 degrees and decrease the value of our X-direction to get back to -90. The Y- and Z- direction of the orientation (Pitch and Roll) are kept at zero. During the whole simulation there will be a forwards speed of 1 unit/s.



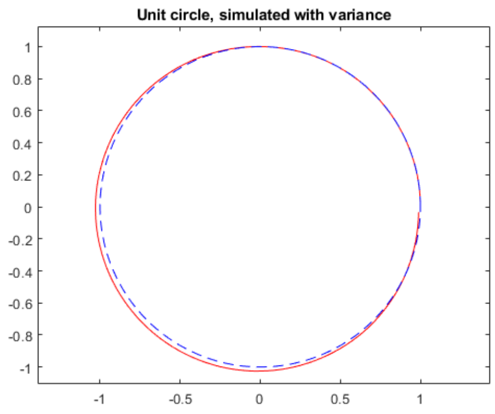
(a) The path visualized

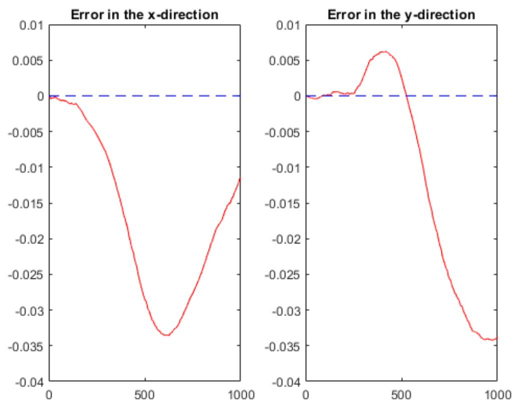


(b) The error in the x and y direction (RMSE: 0,0938)

Figure 5.6: Unit circle, simulated without variance

In Figure (5.6) the path of the unit circle is shown. Compared to the unit square the error seems to be much more smooth and continuous. When matching the graph with that of Figure (5.7), the addition of variance is more clearly disturbing the path, resulting in higher discrepancy. When calculating the RMSE of these paths for the circle without variance and with the variance implemented we get 0,0938 and 0,1502, respectively.

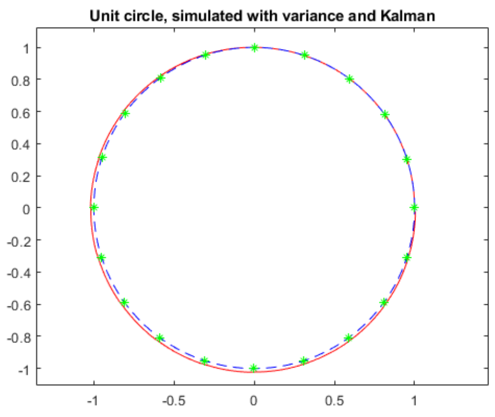




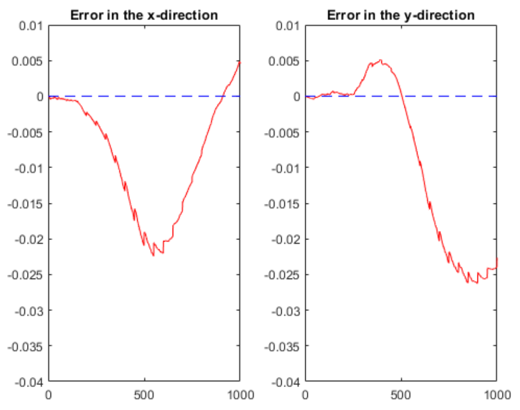
(a) The error in the x and y direction (RMSE: 0,1502

Figure 5.7: Unit circle, simulated with variance

The addition of the Kalman filter helps a bit more in the case of the circle than in the square. If we use the same uncertainties as described previously, we end up with an RMSE of 0,1243.



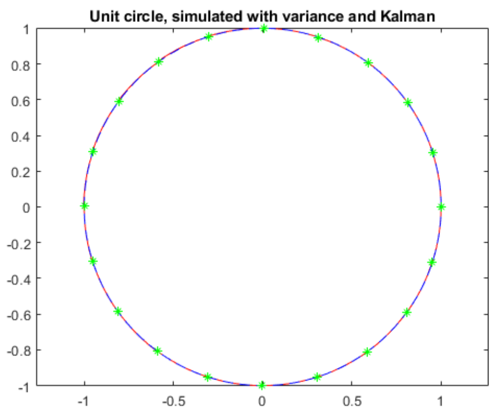
(a) The path visualized



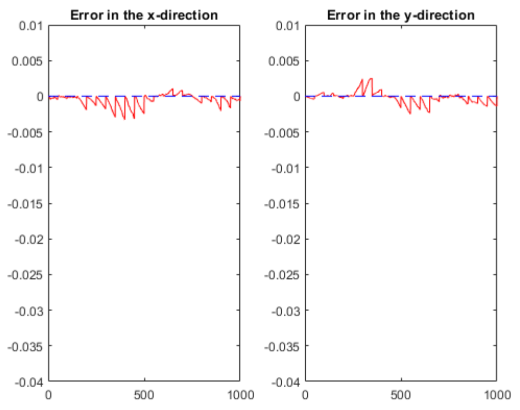
(b) The error in the x and y direction (RMSE: 0,1243)

Figure 5.8: Unit circle, simulated with variance and Kalman filter

We have seen that setting the measurement uncertainty to zero helps our algorithm to stay closer to the true path. When we do this with the circle it is shown to us again: The error margin stays much closer to zero and this leads to a lower RMSE, namely 0,0307.



(a) The path visualized

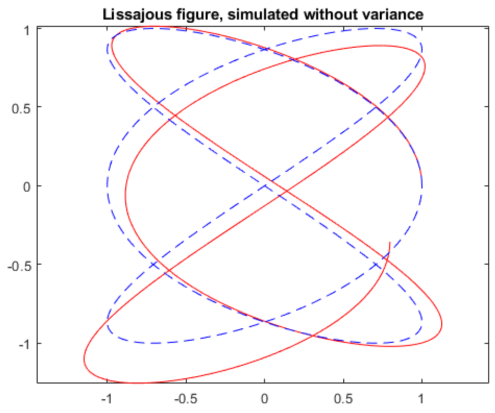


(b) The error in the x and y direction (RMSE:0,0307)

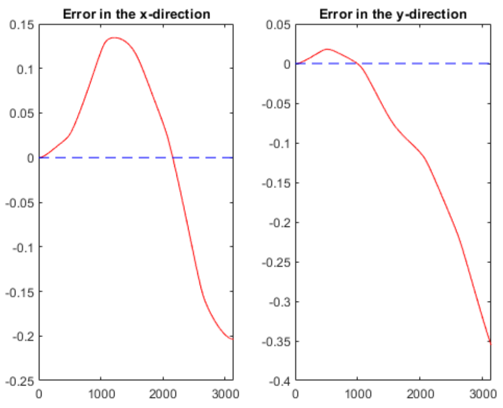
Figure 5.9: Unit circle, simulated with variance and Kalman filter (Measurement uncertainty = 0)

5.3.3. Lissajous figure

The last figure that will be simulated is a *Lissajous figure*. This one is created using the parametric expression of $(x(t), y(t)) = (cos(3t), sin(3t))$. This is the hardest one to simulate, by far. When walking along the figure, not just the orientation changes(this is just the derivative of the parameterisation), but also the speed of the movement is changing. In the end an estimation was found (see Appendix D). This is shown below.



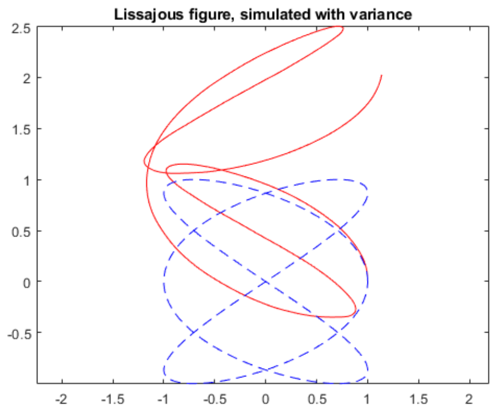
(a) The path visualized



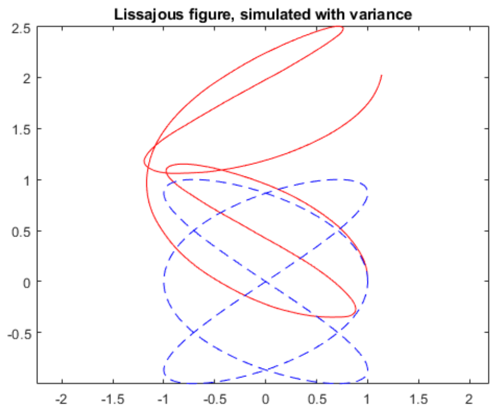
(b) The error in the x and y direction (RMSE:0,328)

Figure 5.10: Lissajous figure, simulated without variance

Looking at Figure (5.10) and (5.11), it is clear to see that this is way harder to fit than the previous 2 examples. This can be quantified when looking at the RMSE of these simulations. Without variance, the average deviation is 33 cm and with variance it is more than 1 meter. This is quite sizable considering the total path length is only just over 15 meter.

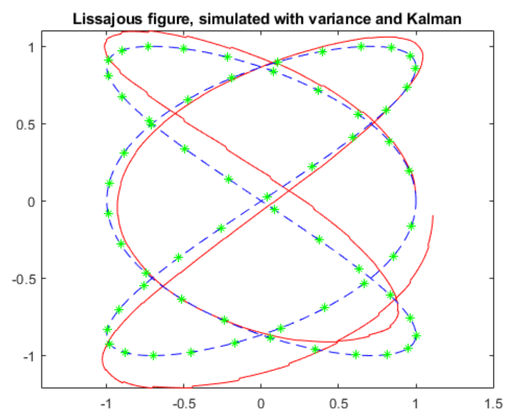


(a) The path visualized

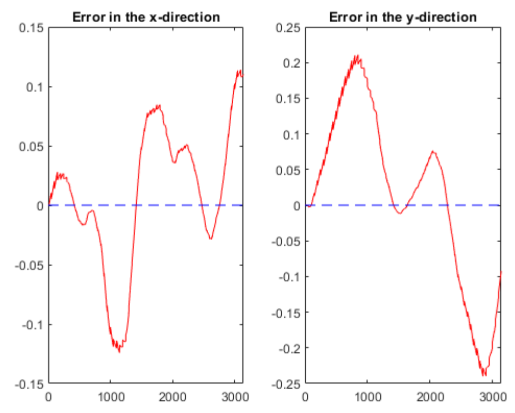


(b) The error in the x and y direction (RMSE:1,0381)

Figure 5.11: Lissajous figure, simulated with variance



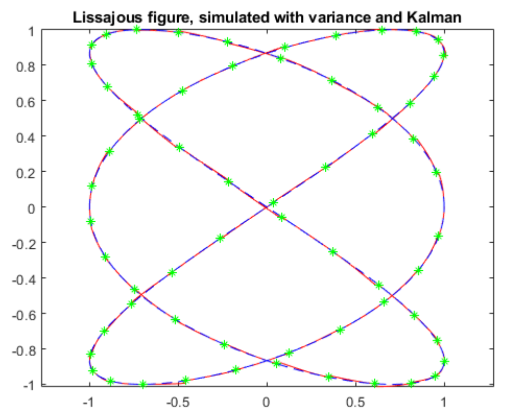
(a) The path visualized



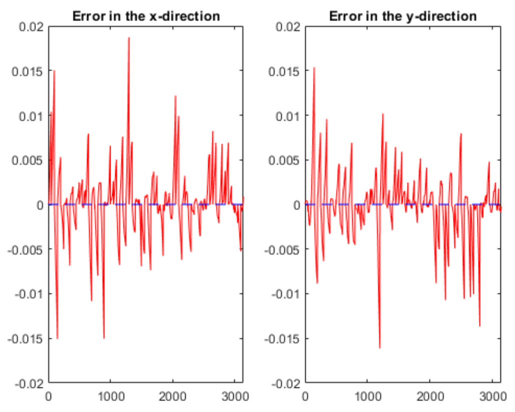
(b) The error in the x and y direction (RMSE:0,3456)

Figure 5.12: Lissajous figure, simulated with variance and Kalman

Luckily, the Kalman filter can help us. When we add the filter the RMSE drops to 0,34 meter and when we remove the measurement uncertainty once more, the RMSE is just 5 centimeter. The lissajous figure really helps to illustrate the difficulties when working with accelerations and rotations, but it also presents the strength of the Kalman filter.



(a) The path visualized



(b) The error in the x and y direction (RMSE:0,0592)

Figure 5.13: Lissajous figure, simulated with variance and Kalman (Measurement uncertainty = 0)

5.3.4. Conclusion of the simulation

After doing these 3 simulations some things have become clear. Starting with the variance, the addition of variance is creating some serious difficulties for the algorithm. Although the variances are only given to the acceleration, and their values seemed very low when characterizing the sensors in Section 2, it still increases the RMSE by a sizable amount in all 3 cases. In these examples the difference between 6 and 10 centimeters might seem negligible, but one needs to consider that these are very short distances. If the distance of the path were to increase, we expect a bigger error as well.

Secondly, the Kalman filter is always providing us with a better fit than without the filter. Even if it does not make up for the increase in RMSE from the variance it has always helped here, and seems to help more on bigger, complicated figures such as the lissajous figure.

Lastly, an interesting discovery was made. Setting the measurement uncertainty(R) to zero has decreased the RMSE to very small margins. This is justified in these simulations, but may not apply to the real-world experimental data. This has caused the following experiments to be modelled with both an algorithm with R=1 and R=0 in the next chapter.

RMSE Values	Square	Circle	Lissajous figure
No Variance	0,0639	0,0938	0,328
With Variance	0,1090	0,1502	1,0381
Difference to no variance %	+70%	+60%	+216%
With Kalman filter	0,0924	0,1243	0,3456
Difference to with variance%	-15%	-17%	-77%
No measurement uncertainty	0,0303	0,0307	0,0592
Difference to with variance%	-72%	-75%	-94%

6
The experimental data

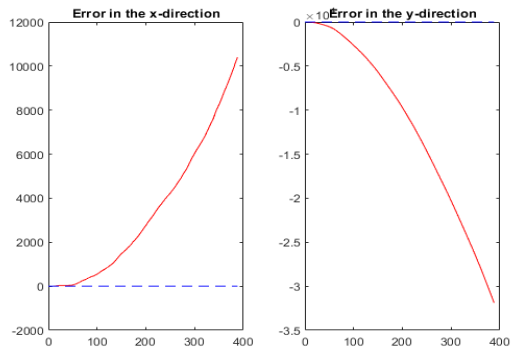
The initial goal of this research was to explore the possibility of using an INS for navigation instead of GPS. So far, a model was set up and the results on simulated data give some sort of confidence into the calculations. The next step is to test the algorithm with real-world experimental data. Again, we test using 3 different scenario's: walking in a straight line for about 500 meter, walking a square(with a total perimeter of 500 meter as well), and lastly walking a couple of circles around a roundabout. For every case a graph without the Kalman algorithm is shown, and a graph of the algorithm with the filter, showcasing its sensitive behaviour. The other graphs will be given in appendix A.

6.1. Straight line

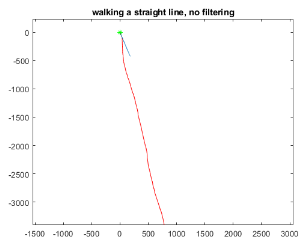
The most simplistic scenario is that of walking a straight line. In this experiment the INS was kept in a flat, forward-facing position where there was an attempt to keep the orientation in the same direction at all times. At the start the INS was held still, then the data collection was started, then the walk begun.

6.1.1. Without Kalman filter

As we have shown in the simulations, the algorithm works quite well if it is given the perfect input in terms of acceleration and orientation data. Yet, if we look at Figure (6.1), the algorithm without Kalman filter is not performing very well. The predictions tell us that we are moving much faster than the GPS is registering and also in a different direction than expected. On a positive note, the estimations are in a straight line, thus the orientation is stable.



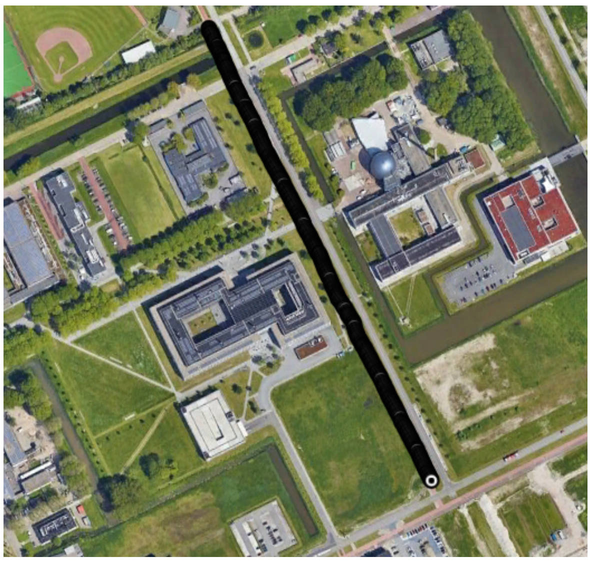
(a) The path visualized



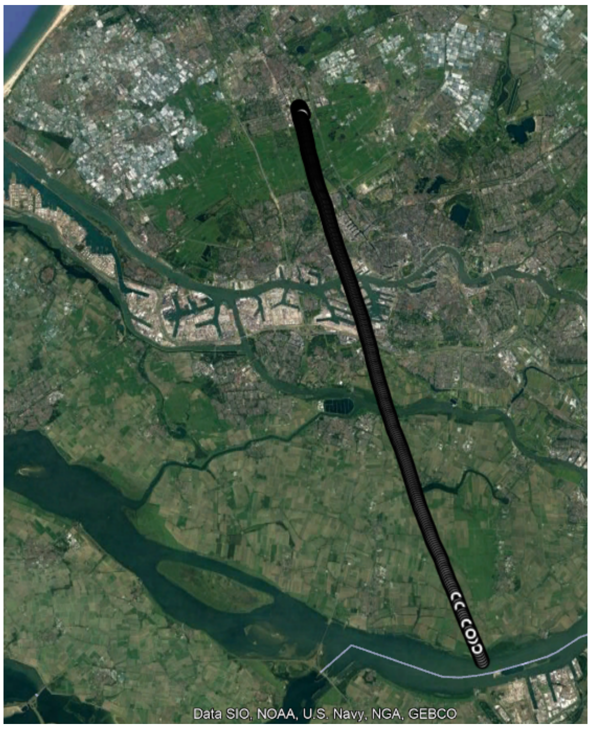
(b) The predicted path

Figure 6.1: Unit circle, simulated with variance and Kalman filter (Measurement uncertainty = 0)

If we show the figure on top of a map we see the total scale of the error, instead of walking 500 meter in Delft, the system thinks we have travelled almost 30 kilometers, all the way to the Haringvliet. Clearly, some sort of filtering is necessary



(a) The path visualized



(b) The predicted path

Figure 6.2: Unit circle, simulated with variance and Kalman filter (Measurement uncertainty = 0)

6.1.2. With Kalman filter

In this section, the filtering will be done in 6 different ways. The simulations have shown that setting the measurement uncertainty R to zero instead of one could be beneficial. Therefore we will be looking at both cases. Also we will look at the difference in using the GPS measurements every 1, 3, or 5 seconds.

In the table a quick overview is given of the resulting RMSE. This RMSE is calculated using the GPS data, thus it indicates the average distance from the nearest GPS point during the experiment in meters. IF the algorithm were to be updating every second, we recognise the benefit of setting the measurement uncertainty to zero. this gives almost no error, whereas the case with R=1, there is an average error of 15 meter. However, when we look at the higher timeframes, where the Kalman filter only operates every 3 or 5 seconds, the case with more measurement uncertainty is outperforming the latter. This could be explained by the algorithm 'overcompensating' for the mistakes in the estimation. It could therefore change the course or speed of the state too much and cause it to give a wrong estimation in the next step once more.

	Seconds between GPS		
	1	3	5
R = 1	15	194	427
R = 0	2.3e-17	6.9e6	1.7e32

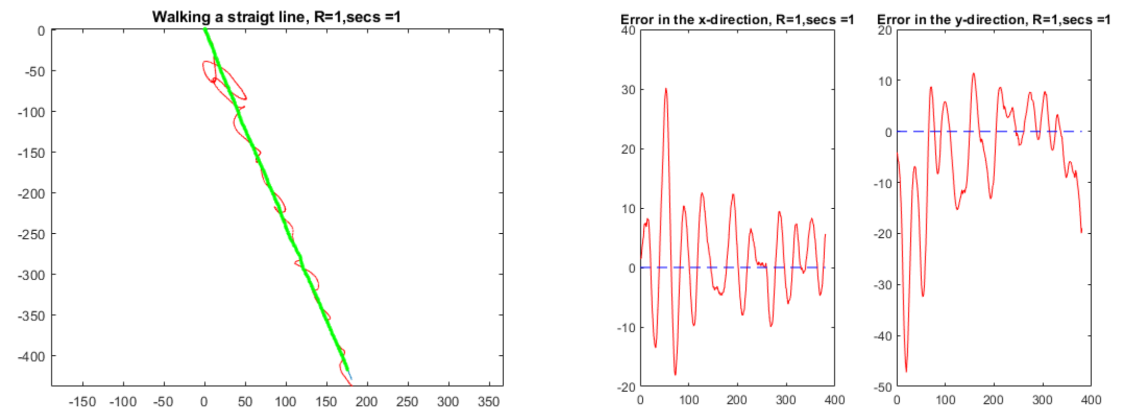


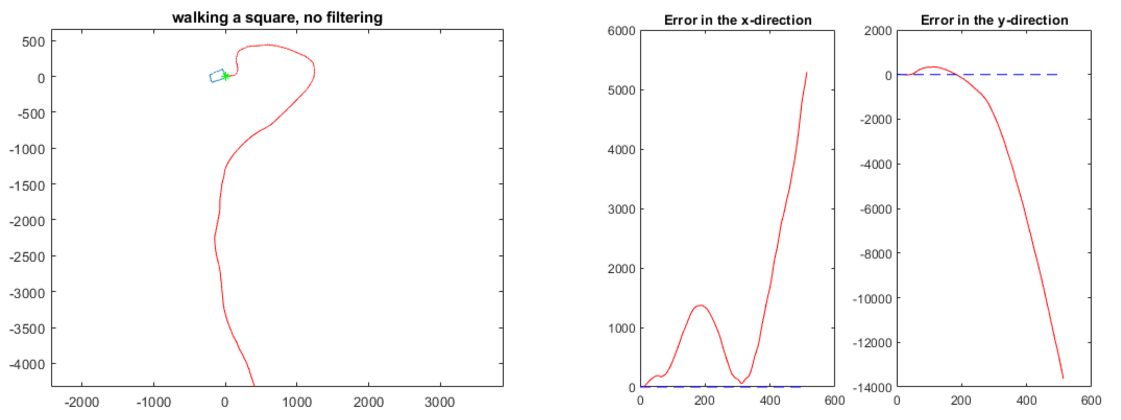
Figure 6.3: When the measurement uncertainty is 1 meter, but we filter every second, the prediction stays close to the real path

6.2. A square

For the experiment of walking a square, the goal was to attempt to mimic the behaviour of the computer simulation as done in the previous chapter. This means that the tablet was held flat, but always pointing the same direction. In contrary to the other 2 experiments this involved some awkward handmovements to keep the tablet in the same direction instead of it pointing at the direction we're moving.

6.2.1. Without Kalman filter

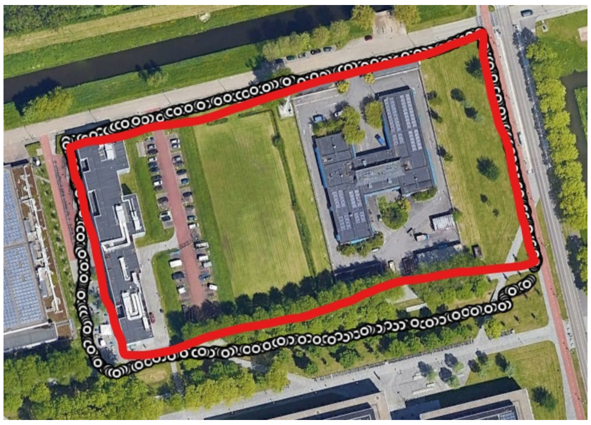
In Figure (6.4) the paths are shown on top of a map. The figure on the left is the GPS data. We see an example of how the GPS is not perfectly accurate here. The actual route that was taken is shown in red. Nevertheless, the dead-reckoning algorithm without filtering is more deceitful. In this case the direction is not maintained in a good manner and the final shape of our route looks nothing like the square from the actual path.



(a) The path visualized

(b) The predicted path

Figure 6.4: Unit circle, simulated with variance and Kalman filter (Measurement uncertainty = 0)


(a) The path visualized

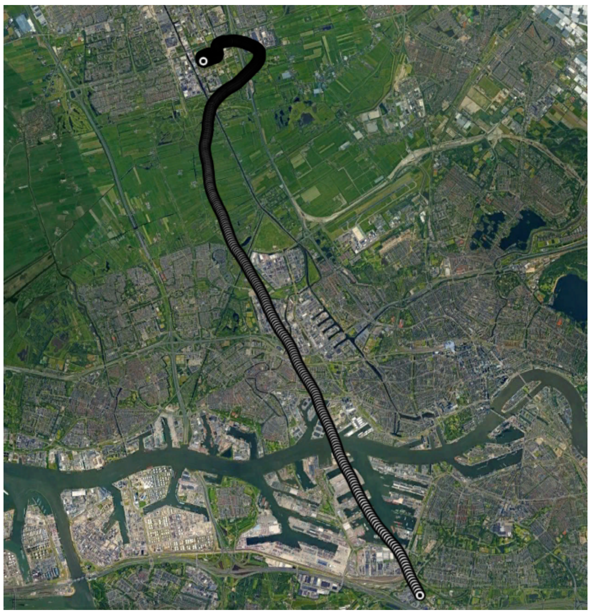

(b) The predicted path

Figure 6.5: Unit circle, simulated with variance and Kalman filter (Measurement uncertainty = 0)

6.2.2. With Kalman filter

The results from experimenting with the Kalman filter are actually quite similar to the experiment with the straight line. As the interval of GPS connections increases, the error value increases as well. The removal of measurement uncertainties is only really viable if the measurements are given every second, otherwise the error becomes immense

	Seconds between GPS		
	1	3	5
R = 1	15	133	388
R = 0	2.8e-17	7.9e9	1.3e47

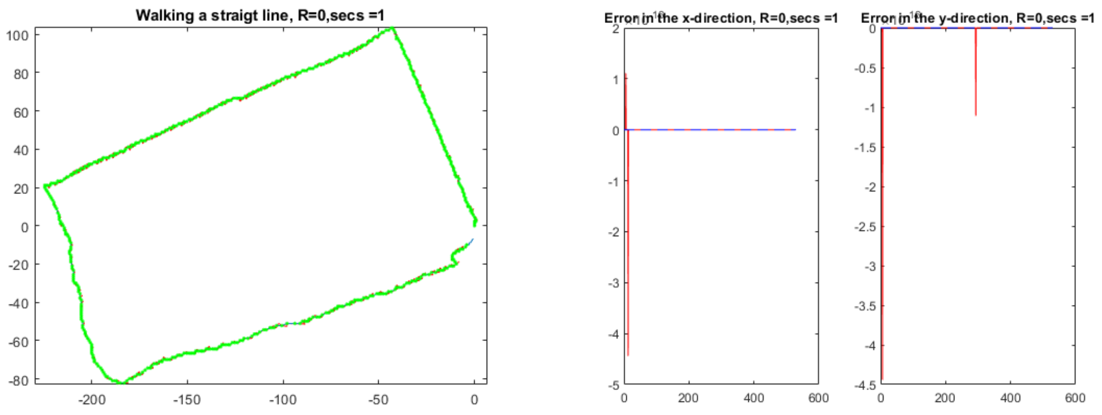


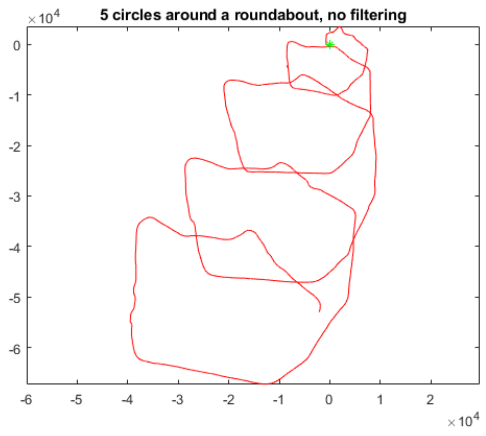
Figure 6.6: When R=0 and we update every second, the resulting path (in red) is almost exactly the GPS data (in green)

6.3. Circles around a roundabout

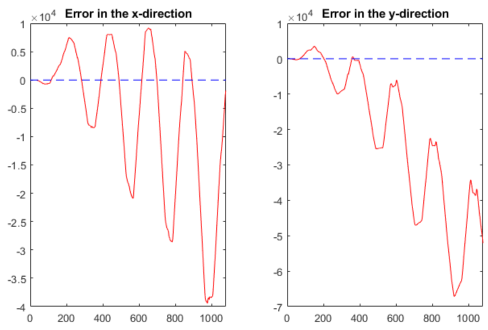
For the circles around the roundabout the INS was placed in a horizontal position and facing forwards in the direction of the movement at all time. This is similar to the simulated data of the circle. The roundabout has a circumference of just under 200 meters and the experiment consisted of about 5 rounds. This gives a total distance of just under 1000 meter.

6.3.1. Without Kalman filter

When we apply the algorithm without the Kalman filter we see that we are dealing with a lot of integration drift.



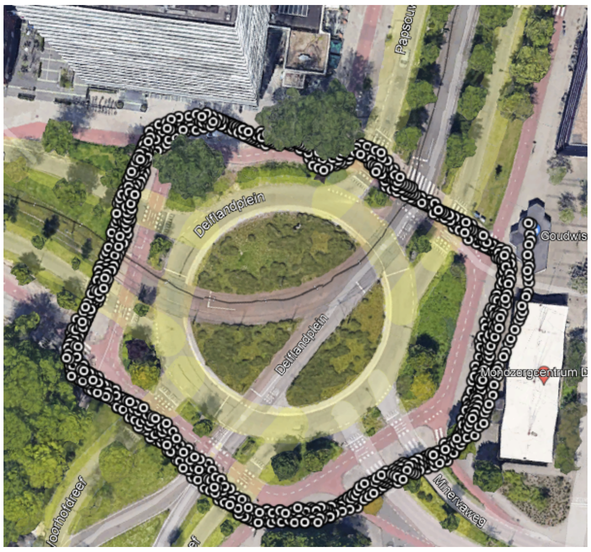
(a) The path visualized



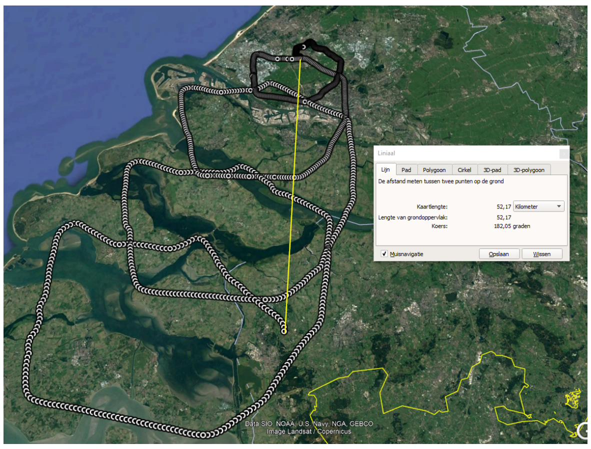
(b) The error in the x and y direction (RMSE:3.2591e+04)

Figure 6.7: Unit circle, simulated with variance and Kalman filter (Measurement uncertainty = 0)

We see that the error is sizable, as the RMSE is more than 30.000 units, but to visualize it more clearly, the points have been plotted out on top of a satellite picture of the area.



(a) The path visualized



Liniaal

Lijn | Pad | Polygoon | Cirkel | 3D-pad | 3D-polygoon

De afstand meten tussen twee punten op de grond

Kaartlengte: 52,17 Kilometer

Lengte van grondoppervlak: 52,17

Koers: 182,05 graden

✓ Muisnavigatie

Opslaan Wissen

Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image Landsat / Copernicus

(b) The predicted path

Figure 6.8: Unit circle, simulated with variance and Kalman filter (Measurement uncertainty = 0)

On the left side of the picture, the GPS data is shown. As mentioned previously it shows some fairly consistent data of walking around the roundabout. On the right side however, the predicted path without filtering is shown. Every 'lap' around the roundabout translates into a bigger ellipse in the predicted path. It shows some clear drift to the south-west as well. In just 15 minutes time, the algorithms has drifted the estimated location all the way from Delft to Belgium almost. After 5 rounds, the error is about 52 km.

6.3.2. With Kalman filter

In this final experiment, the distance was larger and the path is visited multiple times. This gives some interesting results, likewise when we extend the algorithm with the Kalman filter. Most notably, the values are much bigger here. Apart from the case where R=0 and there is 1 second between each GPS signal, all tests have resulted in an average error margin of more than a million kilometers. In the worst case, the average distance is 2.0e106, whereas the diameter of the sizable universe is 1.8e26 meter [13].

	Seconds between GPS		
	1	3	5
R = 1	2.7e9	1.6e16	2.5e11
R = 0	4.1e-17	1.1e89	2.0e106

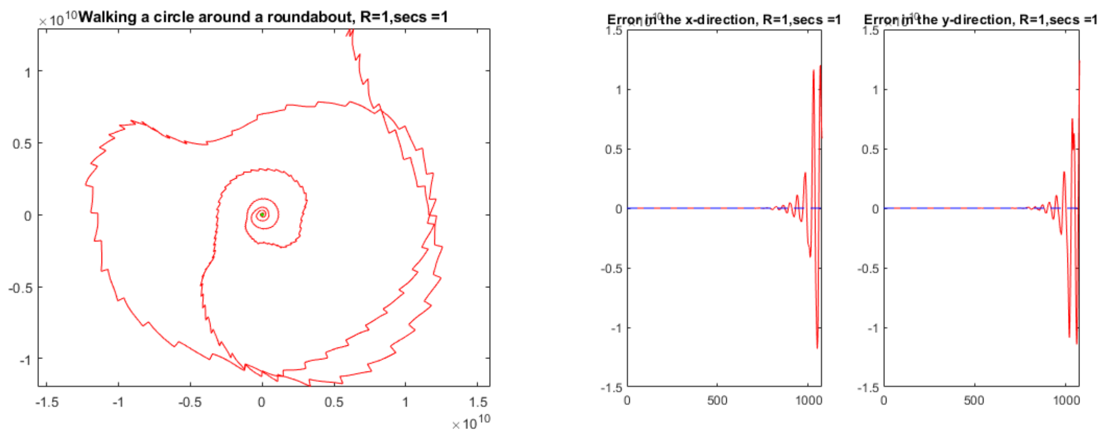


Figure 6.9: Even when we update every second, the experiment with the roundabout will still be off if R=1

7
Conclusion

In this report the possibilities of navigating using an INS was investigated. The goal was to see if there is an opportunity to be less reliable on GPS, or to obtain another method which could help GPS in its errors. The scope of this report is limited to the usage of a tablet and its accelerometer and gyroscope sensor data. We have shown that it is possible to create an algorithm that uses dead-reckoning and the Kalman filter to simulate some promising results. There might certainly be potential in navigation using Inertial Navigation Systems.

Despite this observation, testing in the real world showed some serious deviation from the actual travel. The only way that the location estimated using the INS was when comparing it with the GPS every second. This meant that the predicted value shifted towards the GPS data every second. When losing GPS data for longer periods of time, the algorithm starts to drift further and further away from the true location, even on short trips or time intervals.

Nonetheless, this research has given a beginning for more research towards the modelling of INS navigation. A lot has been learned and much fallacies have been considered. The main conclusion from this report would be that however physical theorems and computational models work in theory, the real world is a whole other scale of complexity. Sometimes things are just way harder than you would expect.

Further research can be done in multiple ways. First of all, the data could be gathered using better hardware for the accelerometer and gyroscope. This would be more precise or have a higher frequency of data collection. Using more sensors, such as an magnetometer would also be interesting. Secondly, new ways of designing the algorithm have yet to be explored. Not just tweaking or calculating the uncertainties could result in better outcomes. Likewise, the usage of dead-reckoning could be replaced by other numerical methods, like Runge-Kutta. Lastly, the way of testing could be much more rigorous. Multiple simulations could be done with a range of parameters, shapes and lengths. The tests outside could at first be done with more stable objects like drones or cars, which do not experience many small changes to the direction and acceleration, while also being able to operate on a larger scale by moving faster.

8 Discussion

In this thesis, an algorithm was designed aimed at calculating the location of an INS. In the process of creating this algorithm, some assumptions had to be made. For instance, the z-coordinate is not taken into account most of the time. Also, other occurrences from physics were not considered. Think of things like the Coriolis-effect or the curvature of the earth.

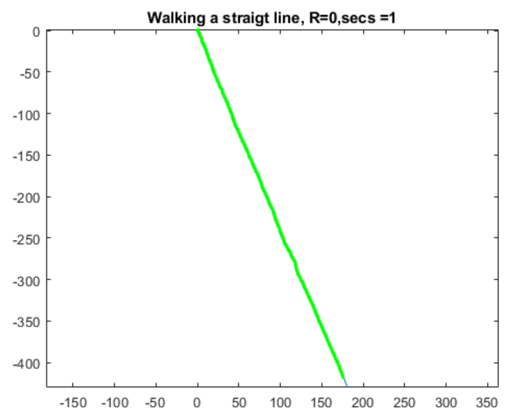
While these factors have been considered negligible, some big assumptions were made by using dead-reckoning as a method. This method expects previous history of the data to have no play in the next prediction, thus creating a resemblence of a random walk. This might explain why the mean of the acceleration data is not equal to zero, even though the INS was standing still before starting and finishing the experiments.

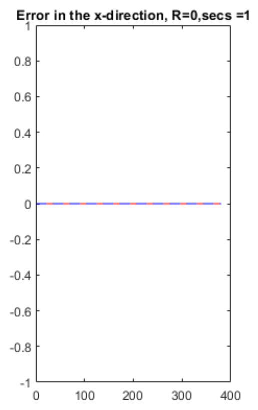
These experiments were not done properly all the time either, the hardware gave some faulty data, sometimes returning NaN, there were too few experiments to exclude chance in the mix and the conditions were not always very similar. Different locations, different dates and a slightly different algorithm are not ideal.

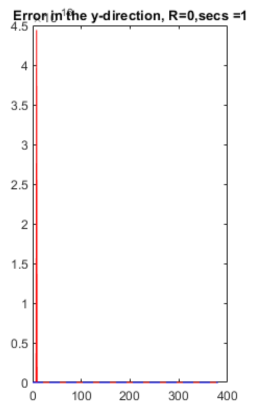
Lastly, the algorithm uses a number of uncertainties to apply the Kalman filter to the system. These values have been estimated on a basis of other sources and guessing. This resulted in very different outcomes as showcased with the measurement uncertainty R, which value to take is still to be determined.

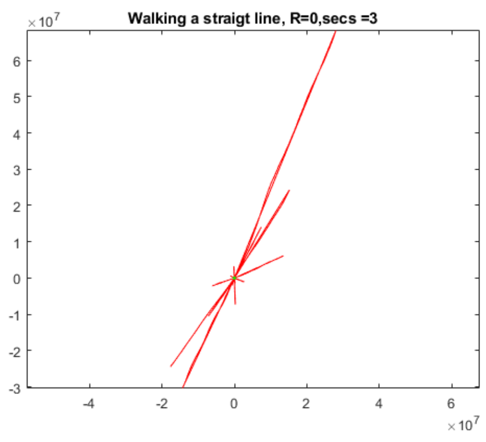
A
Extra graphs from the experiments

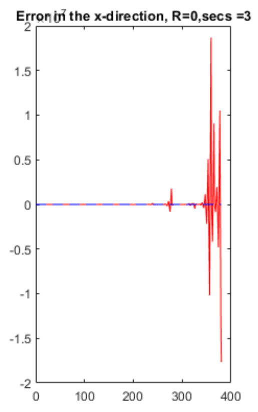
A.1. Straight line

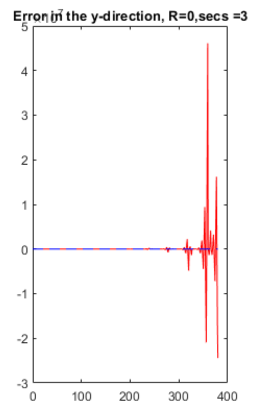


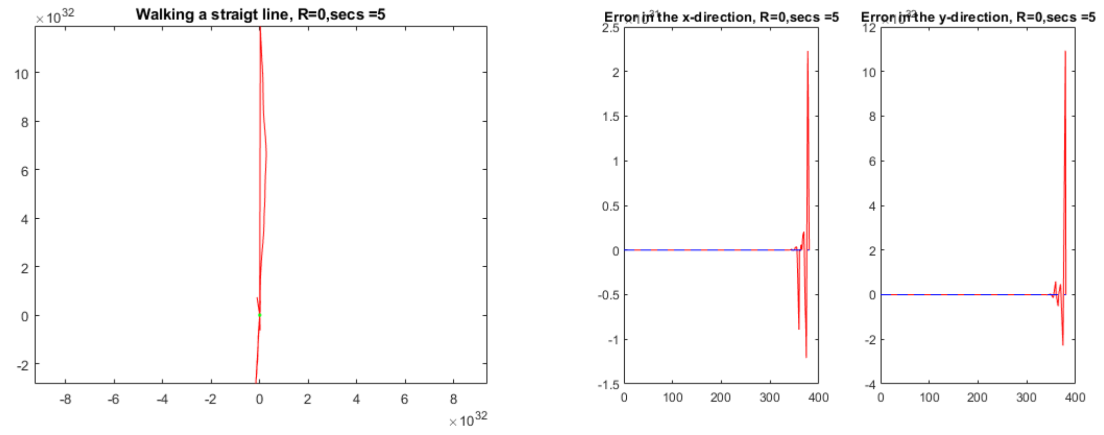


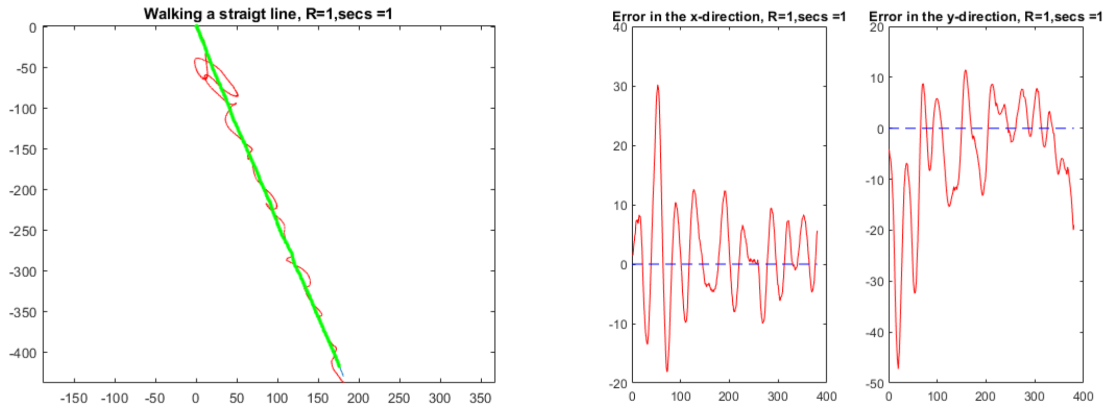


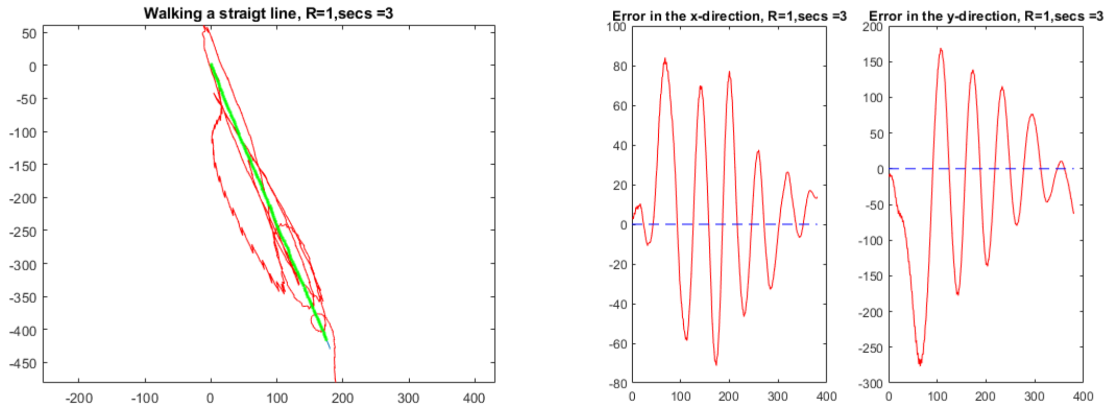


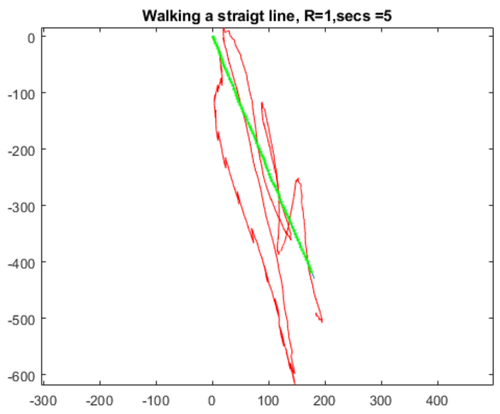


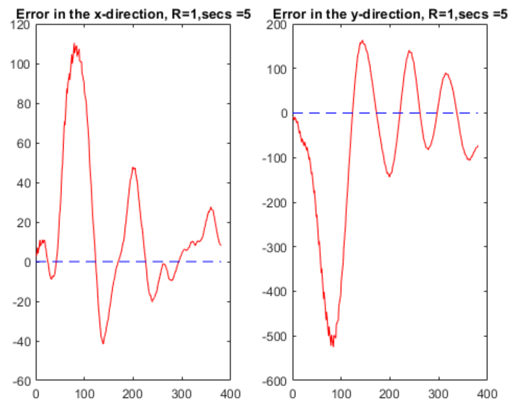



Walking a straigt line, R=0,secs =5
× 10^32
× 10^32
Error in the x-direction, R=0,secs =5
Error in the y-direction, R=0,secs =5

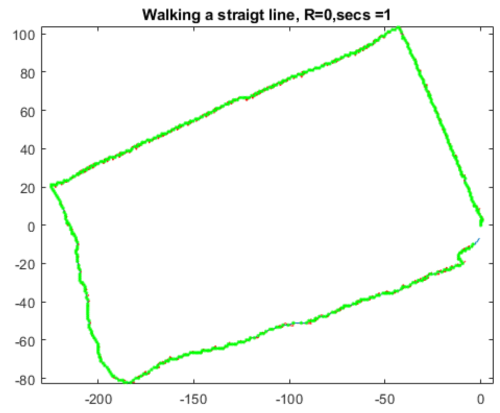

Walking a straigt line, R=1,secs =1
Error in the x-direction, R=1,secs =1
Error in the y-direction, R=1,secs =1

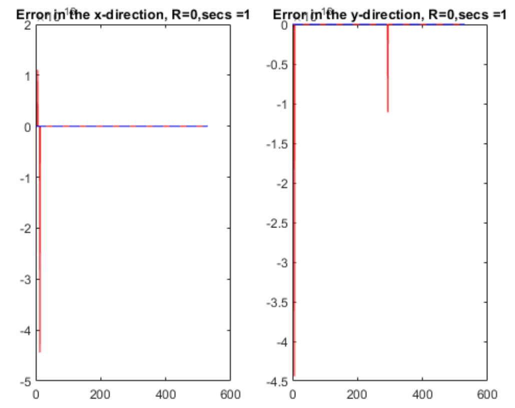

Walking a straigt line, R=1,secs =3
Error in the x-direction, R=1,secs =3
Error in the y-direction, R=1,secs =3

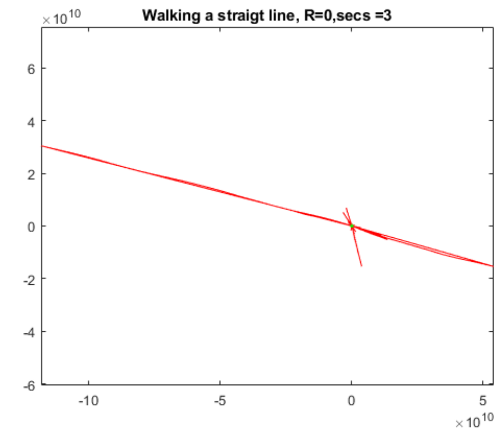

Walking a straigt line, R=1,secs =5

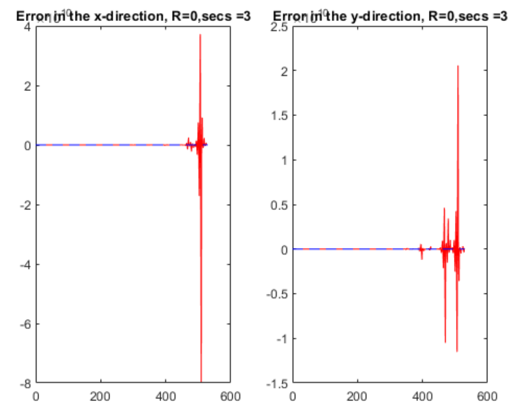

Error in the x-direction, R=1,secs =5
Error in the y-direction, R=1,secs =5

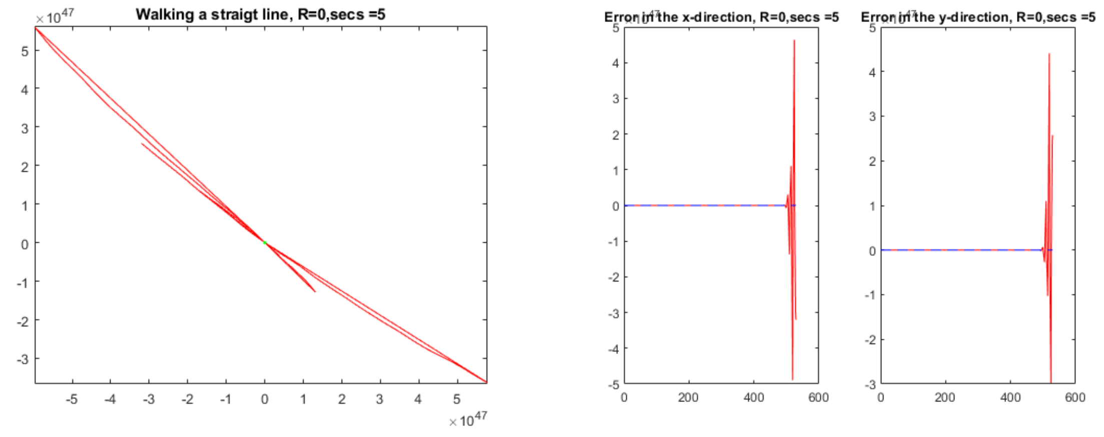
A.2. Square

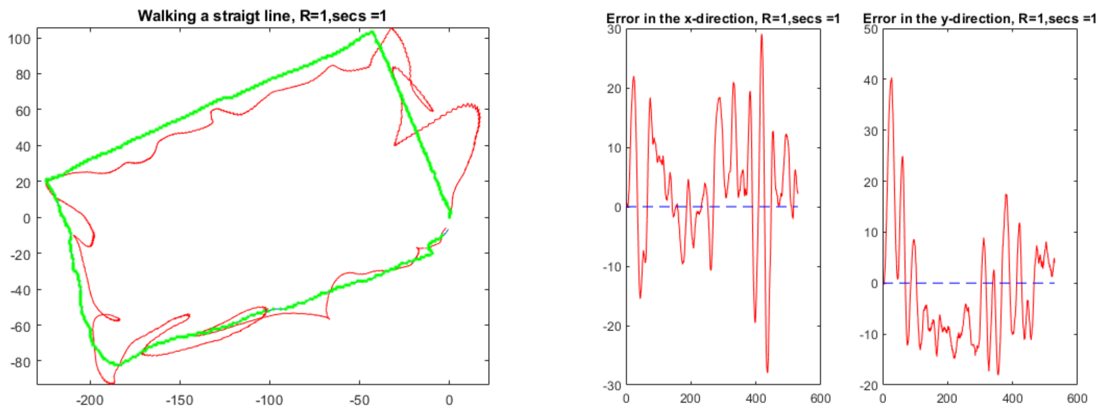

Walking a straigt line, R=0,secs =1

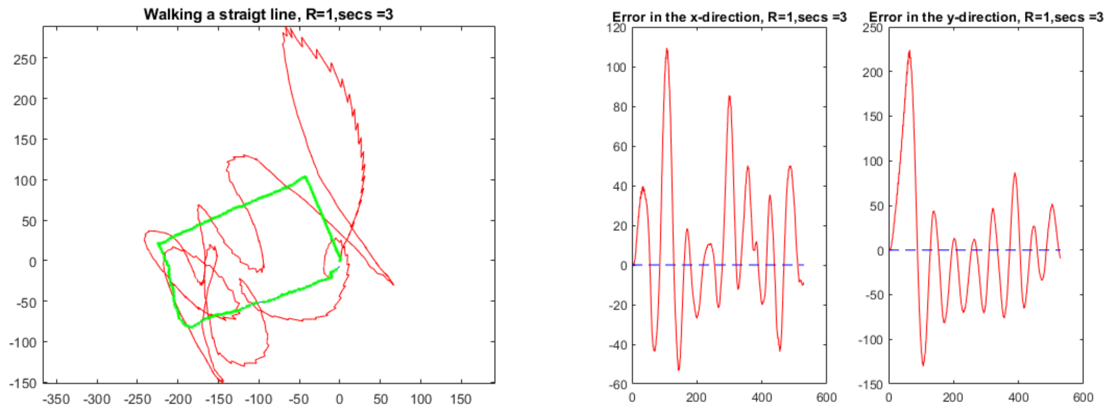

Error in the x-direction, R=0,secs =1
Error in the y-direction, R=0,secs =1

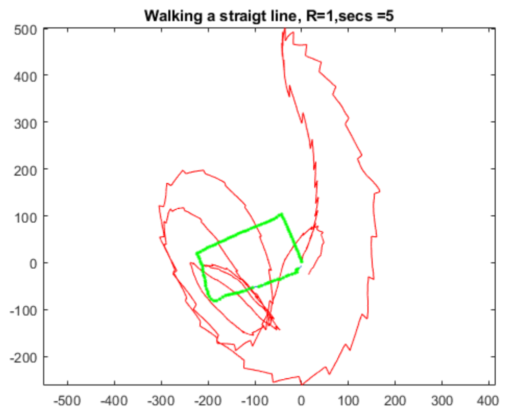

Walking a straigt line, R=0,secs =3

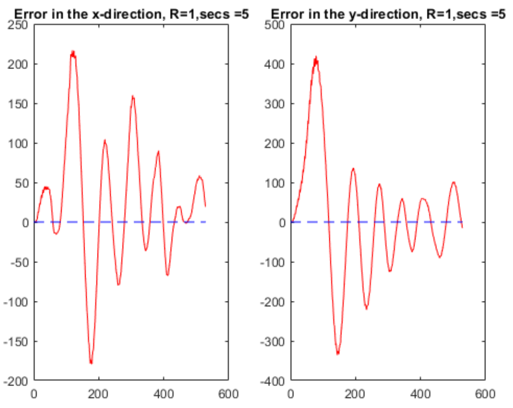

Error in the x-direction, R=0,secs =3
Error in the y-direction, R=0,secs =3


Walking a straigt line, R=0,secs =5
×10^47
×10^47
Error in the x-direction, R=0,secs =5
Error in the y-direction, R=0,secs =5

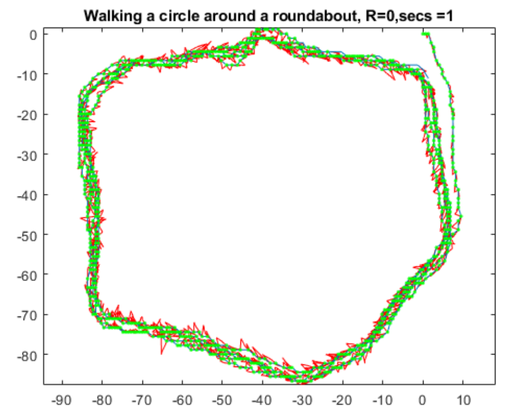

Walking a straigt line, R=1,secs =1
Error in the x-direction, R=1,secs =1
Error in the y-direction, R=1,secs =1

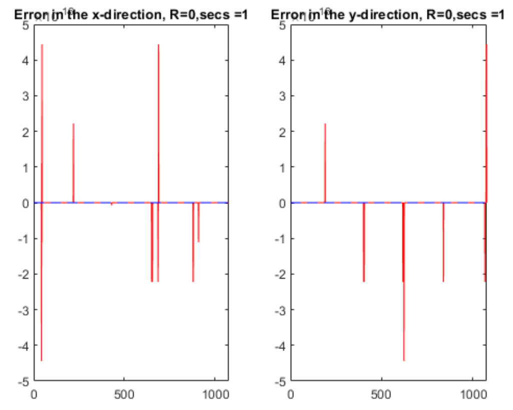

Walking a straigt line, R=1,secs =3
Error in the x-direction, R=1,secs =3
Error in the y-direction, R=1,secs =3

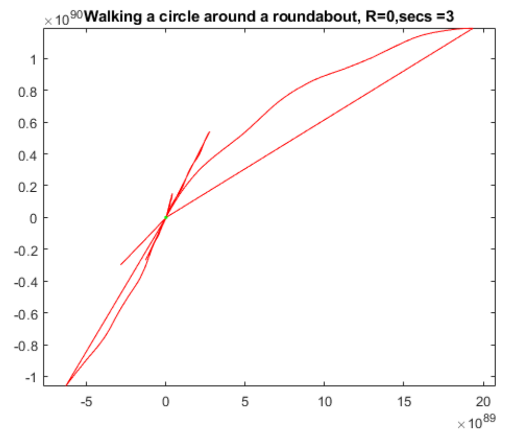


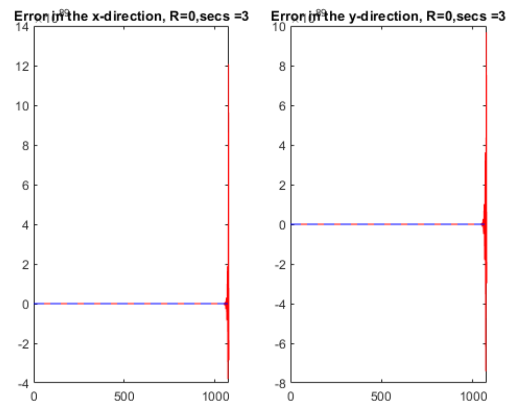


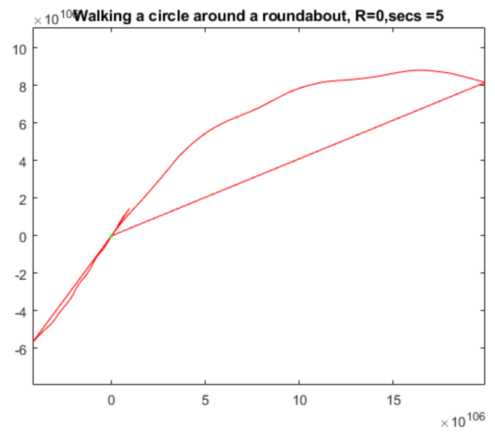
A.3. Roundabout

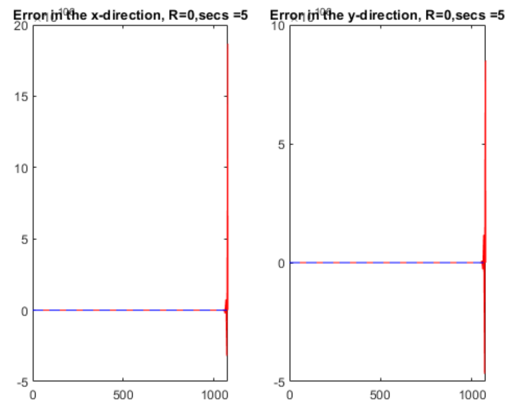


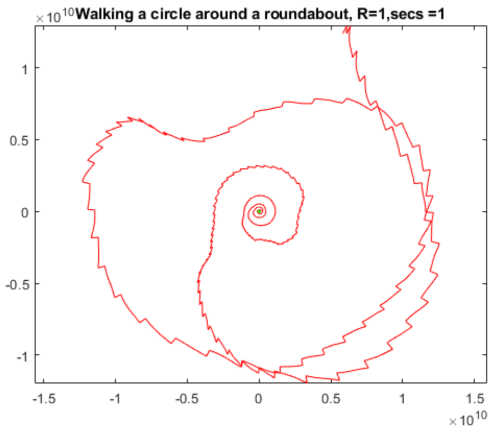


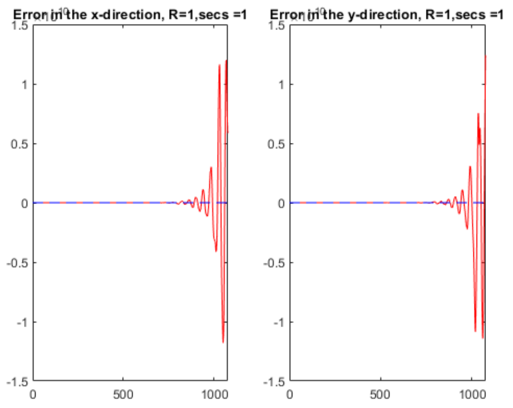


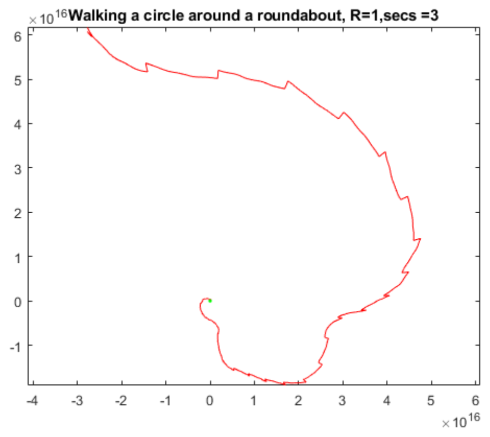


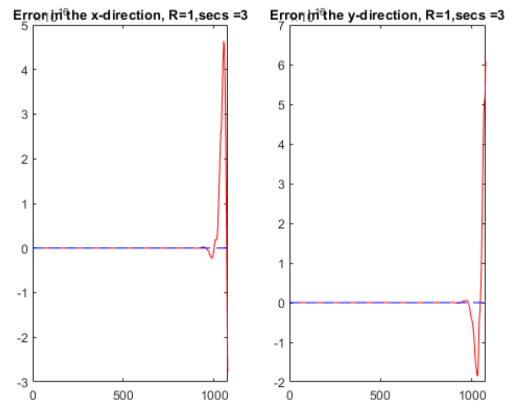

Walking a circle around a roundabout, R=0,secs =5
×10^106

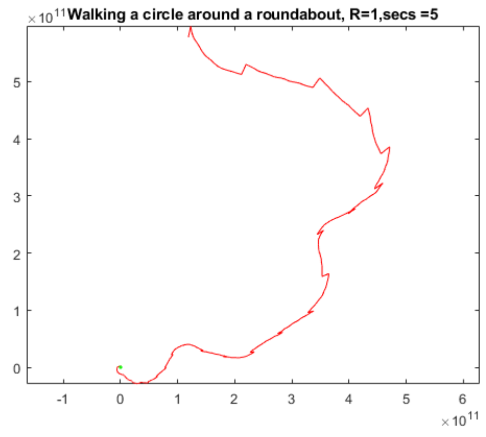

Error in the x-direction, R=0,secs =5
Error in the y-direction, R=0,secs =5

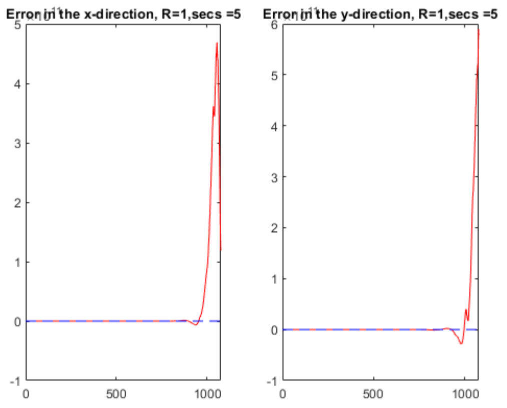

Walking a circle around a roundabout, R=1,secs =1
×10^10


Error in the x-direction, R=1,secs =1
Error in the y-direction, R=1,secs =1


Walking a circle around a roundabout, R=1,secs =3
×10^16


Error in the x-direction, R=1,secs =3
Error in the y-direction, R=1,secs =3


Walking a circle around a roundabout, R=1,secs =5
$\times 10^{11}$
$\times 10^{11}$


Error in the x-direction, R=1,secs =5
Error in the y-direction, R=1,secs =5

B

Matlab code for the simulations

```
rng default

%% CHANGEABLE
len = 1000;
R = 1;
secs=100;
C = 0;

% Kalman matrices
r = 0;
Rn = diag([r r]);
q = 0.001;
Qn = diag([q q q q q q]);
p1 = 1;
p2 = 0.1;
Pn = diag([p1 p1 0 p2 p2 p2]);

%%
vars = C*[2.4572e-05   3.1813e-05   1.6963e-04;
    0.0027   4.4958e-06   1.9561e-05].^(1/2);

means = [-0.0094    0.1601    9.7719;
  124.6551   -0.9315    0.0454];

Acceleration.X = normrnd(means(1,1),vars(1,1),len,1);
Acceleration.Y = normrnd(means(1,2),vars(1,2),len,1);
Acceleration.Z = normrnd(means(1,3),vars(1,3),len,1);
Orientation.X(1:len) = normrnd(0,vars(2,1),len,1);
Orientation.Y(1:len) = normrnd(0,vars(2,2),len,1);
Orientation.Z(1:len) = normrnd(0,vars(2,3),len,1);

PV = zeros(6,length(Acceleration.X));

%%
path = input(['Enter a number for which figure you want to see:' newline ...
    '1 - circle' newline '2 - square' ...
    newline '3 - lissajous' newline]);
switch path
    case 1
```

```
        disp('circle')
        t = linspace(0,2*pi,len);
        fig_x = R*cos(t);
        fig_y = R*sin(t);
        theta = [linspace(-pi/2,-pi,len/4) linspace(pi,-pi/2,3*len/4)];
        PV(1,1) = R;
        T = R*2*pi/len;
    case 2
        disp('square')
        fig_x = [linspace(0,R,len/4) R*ones(1,len/4) linspace(R,0,len/4) zeros(1,len/4)];
        fig_y = [zeros(1,len/4) linspace(0,R,len/4) R*ones(1,len/4) linspace(R,0,len/4)];
        T = 4/len;
        Acceleration.X(1) = Acceleration.X(1) + R*len/4;
        Acceleration.X(len/4)=Acceleration.X(len/4) + R*-len/4;
        Acceleration.X(len/2)=Acceleration.X(len/2) + R*-len/4;
        Acceleration.X(3/4*len)=Acceleration.X(3/4*len) + R*len/4;
        Acceleration.Y(1) = Acceleration.Y(1) + 0;
        Acceleration.Y(len/4)=Acceleration.Y(len/4) + R*len/4;
        Acceleration.Y(len/2)=Acceleration.Y(len/2) + R*-len/4;
        Acceleration.Y(3/4*len)=Acceleration.Y(3/4*len) + R*-len/4;
        theta = zeros(1,len);
    case 3
        disp('lissajous')
        t = linspace(0,2*pi,len);
        fig_x = cos(3*t);
        fig_y = sin(2*t);
        x_a = -3*sin(3*t);
        y_a = 2*cos(2*t);
        x_aa = -9*cos(3*t);
        y_aa = -4*sin(2*t);
        for i = 1:len
            x_a = -3*sin(3*t(i));
            y_a = 2*cos(2*t(i));
            x_aa = -9*cos(3*t(i));
            y_aa = -4*sin(2*t(i));
            abs_v(i) = sqrt((x_a)^2+(y_a)^2);
            abs_aa(i) = (27*sin(3*t(i))*cos(3*t(i))-4*sin(4*t(i)))/sqrt((x_aa)^2+(y_aa)^2);
            theta(i) = -sign(y_a)* acos(x_a/abs_v(i));
            Acceleration.X(i) = Acceleration.X(i) + abs_aa(i);
        end
        Acceleration.X = Acceleration.X + x_aa;
        Acceleration.Y = Acceleration.Y + y_aa;
        Acceleration.Z = Acceleration.Z;
        PV(1,1) = 1;
        T = 2*pi/1000;
end

Orientation.X(1:len) = pi/180*Orientation.X(1:len) + theta(1:len);

if path ~= 2
    for i = 1:len
        % Estimating
        p = Orientation.X(i);
        v = pi*Orientation.Y(i)/180;
        t = pi*Orientation.Z(i)/180;
        R = [cos(p)*cos(v)-sin(p)*cos(t)*sin(v) cos(p)*sin(v)+sin(p)*cos(t)*cos(v) sin(p)*sin(t);
            -sin(p)*cos(v)-cos(p)*cos(t)*sin(v) -sin(p)*sin(v)+cos(p)*cos(t)*cos(v) cos(p)*sin(t);
            sin(t)*sin(v) -sin(t)*cos(v) cos(t)];

        temp = R\[Acceleration.X(i);Acceleration.Y(i);Acceleration.Z(i)];
        if path ==1
            PV(4,1) = 1;
            Acceleration.X(i) = temp(1);
            Acceleration.Y(i) = temp(2);
            Acceleration.Z(i) = temp(3);
        else
            Acceleration.X(i) = Acceleration.X(i) + temp(1);
            Acceleration.Y(i) = temp(2);
            Acceleration.Z(i) = temp(3);
        end
    end

end
acc = [Acceleration.X Acceleration.Y Acceleration.Z]';
%%

s_GPS = 0;

Hn = [1 0 0 0 0 0;
    0 1 0 0 0 0];
green_points = zeros(2,0);

%

for i = 2:len
    % Estimating
    p = Orientation.X(i);
    v = pi*Orientation.Y(i)/180;
    t = pi*Orientation.Z(i)/180;
    R = [cos(p)*cos(v)-sin(p)*cos(t)*sin(v) cos(p)*sin(v)+sin(p)*cos(t)*cos(v) sin(p)*sin(t);
        -sin(p)*cos(v)-cos(p)*cos(t)*sin(v) -sin(p)*sin(v)+cos(p)*cos(t)*cos(v) cos(p)*sin(t);
        sin(t)*sin(v) -sin(t)*cos(v) cos(t)];

    acc = R*[Acceleration.X(i-1) Acceleration.Y(i-1) Acceleration.Z(i-1)]';%-means(1,:)';

    F = eye(6);
    F(1:3,4:6) = T*R;

    G = [0 0 0;
        0 0 0;
        0 0 0;
        T 0 0;
        0 T 0;
        0 0 T];
    G(1:3,1:3) = (T^2/2)*R;

    w = [(T^2/2)*(means(1,1:3)');
        T*((means(1,1:3)'))];

    %PV(4,i) = abs_a(i);
    PV(:,i) = F*PV(:,i-1) + G*acc - w;

    % KALMAN FILTERING
    if mod(i,50*secs)==0
        try
            %% measuring
            pos_Measured = [fig_x(i);fig_y(i)] + normrnd(0,s_GPS,2,1);
            green_points=[green_points pos_Measured];
            %%
            Zn = pos_Measured;
            % PREDICT
            Pn = F*Pn*F' + Qn;
            % KALMAN GAIN
            Kn = Pn*Hn'*inv(Hn*Pn*Hn'+Rn);
            % UPDATE STATE
            PV(:,i) = PV(:,i)+Kn*(Zn-Hn*PV(:,i));
            % UPDATE COVARIANCE
            Pn = (eye(6)-Kn*Hn)*Pn*(eye(6)-Kn*Hn)' + Kn*Rn*Kn';
        catch ME
            disp("ERROR")
        end
    end
end
MSE = mean(sqrt((PV(1,:)-fig_x).^2+(PV(2,:)-fig_y).^2));

%% plot the results
figure;
subplot(1,2,1)
plot(PV(1,:)-fig_x,'r')
hold on
plot([0,len],[0,0],"b--");
hold off
title("Error in the x-direction")
axis([0 1000 -0.04 0.01])

subplot(1,2,2)
plot(PV(2,:)-fig_y,'r')
hold on
plot([0,len],[0,0],"b--");
hold off
title("Error in the y-direction")
axis([0 1000 -0.04 0.01])

figure;
plot(PV(1,1:len),PV(2,1:len),'r')
title("Unit circle, simulated with variance and Kalman")
axis equal
hold on
plot(fig_x(1:len),fig_y(1:len),'b--')
plot(green_points(1,:),green_points(2,:),'g*')
hold off
axis equal

disp(sqrt(MSE))
```

```
figure;
plot(abs_a)
hold on
plot(abs_aa)
j(1) = 2;
for i = 2:len
    j(i) =  j(i-1)+T*abs_aa(i-1);
end
plot(j)
plot(PV(4,:));
legend("abs_a","abs_aa","j","PV")
hold off
```

C
Matlab code for the experiments

```matlab
len = length(Acceleration.X);
kalman = true; % Determines whether Kalman filtering is done
secs = 5; % How often you want to reconnect to GPS

acc = [Acceleration.X Acceleration.Y Acceleration.Z]';

idx_compare = [];
for i=1:length(Position.longitude)
    [val, idx] = min(abs(seconds(Position.Timestamp(i)-Acceleration.Timestamp)));
    if ~isempty(idx_compare) && idx == idx_compare(end)
        break
    end
    idx_compare = [idx_compare idx];
    %idx is the index in the Acceleration data that has the timestamp closest to the timestamp of the
end
len_compare = length(idx_compare);

pos_y = (Position.latitude-Position.latitude(1))*111000; %converting lat/long to meters, with 0,0 the
pos_x = (Position.longitude-Position.longitude(1))*68000;

vars = [2.4572e-05   3.1813e-05   1.6963e-04;
    0.0027   4.4958e-06   1.9561e-05].^(1/2); %The measured variance of the sensors while laying still

means = [-0.0094    0.1601    9.7719;
  124.6551   -0.9315    0.0454];    %The measured mean (bias) of the sensors while laying still


%% Algorithm
PV = zeros(6,len); % PV will be the state vector, consisting of both the location and speed in 3 direc

% Stationary
PV(:,1) = [0 0 0 0 0 0]; %Inital state

%Kalman Matrices
r = 0;
Rn = diag([r r]); % Measurement uncertainty
q = 0.001;
Qn = diag([q q q q q q]); % Process uncertainty
p1 = 1;
```

```matlab
p2 = 0.01;
Pn = diag([p1 p1 0 p2 p2 p2]); % Estimation uncertainty

Hn = [1 0 0 0 0 0; %Output matrix
    0 1 0 0 0 0];

gps_points = zeros(2,0); % The points of the GPS data that are used for filtering

for i = 2:len
    dT = seconds(Acceleration.Timestamp(i)-Acceleration.Timestamp(i-1)); %Calculate the timestep
    % Estimating
    p = pi*Orientation.X(i-1)/180;
    v = pi*Orientation.Y(i-1)/180;
    t = pi*Orientation.Z(i-1)/180;
    R = [cos(p)*cos(v)-sin(p)*cos(t)*sin(v) cos(p)*sin(v)+sin(p)*cos(t)*cos(v) sin(p)*sin(t);
        -sin(p)*cos(v)-cos(p)*cos(t)*sin(v) -sin(p)*sin(v)+cos(p)*cos(t)*cos(v) cos(p)*sin(t);
        sin(t)*sin(v) -sin(t)*cos(v) cos(t)];

    w = [dT^2/2*R ;
        eye(3)*dT] *means(1,1:3)'; % Removing the error vector (bias)

    F = eye(6);
    F(1:3,4:6) = dT*R; % System dynamics matrix

    G = [0 0 0;
        0 0 0;
        0 0 0;
        dT 0 0;
        0 dT 0;
        0 0 dT];
    G(1:3,1:3) = (dT^2/2)*R; % Input matrix

    PV(:,i) = F*PV(:,i-1) + G*acc(:,i-1) - w; % State extrapolation (estimation)

    %% KALMAN FILTERING

    [y, place] = ismember(i,idx_compare); % Check if we can compare with an GPS point

    if kalman == true && y && mod(place,secs) == 0 % Check that we want to compare to this GPS p
        % MEASURE
        Zn = [pos_x(place); pos_y(place)]; % Measurement data
        gps_points=[gps_points Zn];
        % PREDICT
        Pn = F*Pn*F' + Qn;
        % KALMAN GAIN
        Kn = Pn*Hn'*inv(Hn*Pn*Hn'+Rn);
        % UPDATE STATE
        PV(:,i) = PV(:,i)+Kn*(Zn-Hn*PV(:,i));
        % UPDATE COVARIANCE
        Pn = (eye(6)-Kn*Hn)*Pn*(eye(6)-Kn*Hn)' + Kn*Rn*Kn';
    end
end

%% plot the results
plot(PV(1,1:len),PV(2,1:len),'r')
```

```matlab
hold on
plot(pos_x,pos_y)
plot(gps_points(1,:),gps_points(2,:),'g.')
hold off
title("5 circles around a roundabout, with filtering, R=0,secs = 5")
axis equal



%% plot the results
figure;
subplot(1,2,1)
x_error = PV(1,idx_compare)'-pos_x(1:len_compare);
plot(x_error,'r');
hold on
plot([0,length(idx_compare)],[0,0],"b--");
hold off
title("Error in the x-direction")

subplot(1,2,2)
y_error = PV(2,idx_compare)'-pos_y(1:len_compare);
plot(y_error,'r')
hold on
plot([0,length(idx_compare)],[0,0],"b--");
hold off
title("Error in the y-direction")

total_error = sqrt(x_error.^2 + y_error.^2);
MSE = secs*mean(total_error.^2);
RMSE = sqrt(MSE) % Root Mean Squared Error

%% Formatting locations back to lat/longitude coordinates
% Result_y = PV(2,:)'/111000+Position.latitude(1);
% Result_x = PV(1,:)'/68000+Position.longitude(1);

%Res = [Result_x Result_y];
```

Bibliography

[1] An indoor position-estimation algorithm using smartphone imu sensor data. file:///C:/Users/sande/AppData/Local/Packages/microsoft.windowscommunicationsapps_8wekyb3d8bbwe/LocalState/Files/S0/4213/Attachments/An_Indoor_Position-Estimation_Algorithm_Using_Smartphone_IMU_Sensor_Data[14040].pdf. (Accessed on 06/13/2022).

[2] File:3d gyroscope.png - wikimedia commons. https://commons.wikimedia.org/w/index.php?curid=1244193, . (Accessed on 06/29/2022).

[3] File:motorola xoom - kionix kxtf9-1171.jpg - wikimedia commons. https://commons.wikimedia.org/wiki/File:Motorola_Xoom_-_Kionix_KXTF9-1171.jpg, . (Accessed on 06/14/2022).

[4] Gps.gov: Gps accuracy. https://www.gps.gov/systems/gps/performance/accuracy/. (Accessed on 06/13/2022).

[5] How do smartphones know which is the right side up? » science abc. https://www.scienceabc.com/innovation/smartphones-change-orientation-horizontal-landscape-gravity-sensor-accelerometer.html, . (Accessed on 06/29/2022).

[6] How a smartphone knows up from down (accelerometer) - youtube. https://www.youtube.com/watch?v=KZVgKu6v808, . (Accessed on 06/14/2022).

[7] How to reduce gps data errors on android | mad devs blog. https://maddevs.io/blog/reduce-gps-data-error-on-android-with-kalman-filter-and-accelerometer/#android-virtual-sensors, . (Accessed on 06/13/2022).

[8] Kalman filter - wikipedia. https://en.wikipedia.org/wiki/Kalman_filter. (Accessed on 06/20/2022).

[9] Measure device rotation along x, y, and z axes - simulink - mathworks benelux. https://nl.mathworks.com/help/supportpkg/android/ref/orientation.html. (Accessed on 06/29/2022).

[10] Netherlands latitude and longitude map. https://www.mapsofworld.com/lat_long/netherlands-lat-long.html. (Accessed on 06/13/2022).

[11] Solar system exploration: Science & technology: Technology features: Brief history of gyroscopes. https://web.archive.org/web/20150710113230/http://solarsystem.nasa.gov/scitech/display.cfm?ST_ID=327, . (Accessed on 06/14/2022).

[12] Solar storm warning: Russia hit with radio and gps blackout as nasa records sun ejection | science | news | express.co.uk. https://www.express.co.uk/news/science/1624600/solar-storm-warning-russia-struck-blackouts-m3-class-space-weather-flare, . (Accessed on 06/13/2022).

[13] What are the dimensions of our universe in meters? | socratic. https://socratic.org/questions/what-are-the-dimensions-of-our-universe-in-meters, . (Accessed on 06/29/2022).

[14] What is gps spoofing and why is it a problem? | nordvpn. https://nordvpn.com/blog/gps-spoofing/, . (Accessed on 06/13/2022).

[15] What is mems accelerometer? - s.a.dedar. https://sadedar.com/what-is-mems-accelerometer/, . (Accessed on 06/14/2022).

[16] What is mems accelerometer? - s.a.dedar. https://sadedar.com/what-is-mems-accelerometer/, . (Accessed on 06/14/2022).

[17] Alex Becker. Kalman filter tutorial. `https://www.kalmanfilter.net/default.aspx`. (Accessed on 06/13/2022).

[18] Manon Kok, Jeroen D. Hol, and Thomas B. Schön. Using inertial sensors for position and orientation estimation, 2017.