UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCATRONICA

TESI DI LAUREA MAGISTRALE

GROUND VEHICLE DYNAMICS ESTIMATION BASED ON MULTI-BODY MODELS AND NON-LINEAR STATE OBSERVERS

Relatore: Ch.mo Prof. Alberto Trevisani

Correlatore: Ch.mo Prof. Miguel Ángel Naya Villaverde

Laureando: Edoardo Sinigaglia
1082143-IMC

ANNO ACCADEMICO: 2015-16

ABSTRACT

In this work the design of a multi-body oriented state observer is described. The aim is to estimate mechanical system dynamics, in order to correct errors that are present inevitably in the corresponding multi-body models. This allows using the correct multi-body models to generate virtual sensors. In fact, a multi-body model can provide a lot of information about a mechanical system, but it can diverge from reality due to modelling imperfections. Therefore, the task of the observer designed in this work is to make model reliable. Additionally, in order to fully reach the aim, i. e. using multi-body models as virtual sensors, the observer has been designed to guarantee its real-time applicability.

The proposed observer, named *error*EKF, has been firstly applied to a very simple mechanism: the four-bar linkage one, in order to provide a clearer understanding of observer design and implementation. After that, it has been applied to a quite complex mechanical system, a dune buggy vehicle, available, together with its multi-body simulation model, at the Mechanical Engineering Laboratory of the University of A Coruña.

ACKNOWLEDGEMENTS

If I arrived here, some important people who made it possible have to be mentioned.
Thank you to prof. Alberto Trevisani and prof. Miguel Á. Naya, for giving me the opportunity of carrying out this work. I will never forget my stay at LIM.
Thank you to all the lab staff, especially Amelia and Florian for being so kind with me and for sharing some very nice moments, both Frans, Alberto and Urbano for the chats while eating. Actually, I have to thank Alberto also for the computer support. I am still convinced that some of the things he does are quite magic.
Thank you to Emilio, for supporting me in my work and giving me a lot of advice. Thank you to Javier, for welcoming me so warmly.
Thank you prof. Dario Richiedei, thank you to Ilaria for the remote support from Vicenza.
Thank you to mum and dad, for always loving me, encouraging me, scolding me when it was needed, and especially for believing in me. Most of the credit for reaching my targets is due to them.
Thank you to Francesco and Margherita, for their support, love, for a lot of moments laughing together. I am so glad and proud to be their older brother.
A special thank to Valentina, who is always by my side unconditionally. If I reached this goal, in part is thanks to her.
Thanks you to all my friends, listing them all would be too long, but everyone knows he has a special place in my heart. Thank you to my older friends, the ones who shared with me a lot of moments, experiences and adventures since I was a child. Thank you to the newer friends, thank you to Ferrol ones, for leaving me with a big desire of travelling across Italy, Europe and the ocean to meet them.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 MOTIVATIONS

Automotive branch is worldwide one of the main research drivers in engineering field. In fact, without any effort, everybody can realize the importance of cars in everyday life. In a modern vehicle there are several devices of different nature, most of them electronically controlled: the aim of achieving high performances and low fuel consumption leads to an increment of system complexity.
Another field, not less important than the former, in which automotive investigations work is *safety*. There are many devices which help drivers in case of necessity: typically they act in an active way substituting them in those situations, with the aim of avoiding accidents or other problems deriving from anomalous driving conditions. In 2015, 1.25 million people died for road traffic injuries, which are the main cause of death among people aged 15-29 years. About 50 million people suffered non-fatal injuries, which in some cases led to disability [22].
Some estimations [2] show that about 90% of traffic accidents are due to human errors. This is why the so-called ADAS[1] are being frequently introduced in cars. Their introduction reduced traffic accidents: some investigations [11] affirm that up to 27% of car accidents for vehicle lose of control are avoided.

So, a good and complete knowledge of vehicle behaviour seems to be quite necessary. Some information are obtained through direct measurements, however not all we want to know is measurable. For example, some interesting points of a system could not be easily reachable, some measurements could require too much expensive sensors.
All these points are the reason for the growth of the interest in the research field of *state observers*. They are algorithms which perform the estimation of dynamical system physical quantities otherwise unmeasurable.
Modern cars use several reduced-dimension state observers, which utilize partial models. For example, for the estimation of vehicle orientation – yaw, pitch and roll angles – normally two different models are employed, and another one is used to observe suspension dynamics [10]. Their function is directly related to safety and comfort devices, such as stability and traction or semi-active suspension control.

1 Advanced Driver Assistant Systems

1.2 THESIS CONTRIBUTIONS

The aim of this work is to test a state observer strategy for multi-body systems. Multi-body simulations are a common use in the industry, in order to speed up new product development. Nowadays, low-power consumption and high calculating capacity computers make them possible in real-time. In this way, a multi-body model could run on a device installed on the real mechanical system which it represents, working as a source of *virtual measurements* of its physical quantities.
The problem with this approach is that, in general, a multi-body model can be very accurate in the short term *if the forces are accurately known*, but it will diverge over time. In fact, a model is never capable of fully representing the reality: in the case of multi-body dynamics the main source of uncertainties are the forces, which are quite never entirely known. This is why a state observer is required: the estimation of the mechanical system physical quantities allows correcting errors, drifts and uncertainties in the multi-body models. Doing so, the model could be a reliable source of additional information about the mechanical system.
The Kalman filter, due to its suitability as state observer for stochastic systems, seems to be applicable. However, it was originally formulated for first order, linear and unconstrained models, whereas multi-body ones are, in general, second order, highly non-linear, and constrained systems. Moreover, the algorithm should be efficient, in order to run in real-time. This is why, in spite of the previous studies [8, 33, 27, 24], this is still an open research field. A great contribution recently came from [25].

In this work a state observer for the multi-body system of a ground vehicle is presented. It is based on a modified version of the indirect – also called *error-state* – Kalman filter, which was thought up to be used for multi-body dynamics [30]. The goal is to succeed in estimating position and velocity errors committed by the multi-body model and in correcting them, making the model reliable.
In this work only the navigation problem will be treated. The basic idea is that the observer should require only sensors which can be commonly found in car. However, those sensors are typically few to estimate the entire state of a ground vehicle. Therefore, the filter presented in [30] should be extended in order to fully correct the multi-body model in a successful way.
The presence of a corrected multi-body model leads to an additional advantage: if, for some reasons, some measurements were not available the multi-body model could go on providing some information about mechanical system dynamics, whereas a traditional state observer would malfunction. It could happen, for example, with GPS measurements because of the presence of trees, tunnels or any other

building that can shade satellite signal. Or it could simply be caused by the fault or damage of one of the sensors.

The method has been tested on a *dune buggy*[2] vehicle simulation model, programmed by the research staff of the Mechanical Engineering Laboratory, University of A Coruña, where this work was developed.
It is worth noticing that the choice of using only sensors available on commercial vehicles leads to the formulation of a method which is not only new but also feasible. Evidently, if adequate measurements were available, the estimated variable number could be increased.

1.3 OUTLINE

The reminder of the work is organized as follows:

Chapter 2 recalls some fundamentals of the theory of multi-body dynamics, useful to understand vehicle model. A simple a four bar linkage mechanism model is introduced for sake of clarity. In fact, starting from the description of the full buggy model kinematics and dynamics would be too much heavy, and not easily understandable. Then vehicle model is briefly illustrated, focusing on the parts which are observed.

In chapter 3 state estimation is described starting from some basic concept of linear systems theory. In the same chapter the proposed error-state observer is throughly described, and it is applied to the simple mechanism introduced in chapter 2. This lead to some considerations about filter stability and performances.

Chapter 4 addresses the design of vehicle observer, underlining the trade-offs between algorithm efficiency and estimation accuracy.

In chapter 5, simulation results are presented and commented. In the end, conclusions and possible future improvements are drawn.

2 From now on it will be called simply *buggy*.

2 MULTI-BODY MODEL

2.1 MULTI-BODY DYNAMICS

2.1.1 *General concepts*

Engineers usually refer to *multi-body systems* as mechanical systems made of two or more rigid or flexible bodies joined together through kinematic pairs, which allow relative movement between adjacent bodies [13].
Multi-body dynamics is commonly thought to be born in Joseph-Louis Lagrange's book, *Mécanique Analitique*. In this essay the equations of motion for mechanical systems have been formulated for the first time in a general way [6, 18].
Kinematic and dynamic analyses are combined: the former verifies the capability of a system to move following a planned trajectory, the latter identifies internal and external forces acting on the mechanism. General and systematic algorithm for analysing mechanism are searched, so that to automate the calculation and to have it solved by a computer is possible. Normally the result is a model which is made of differential-algebraic equations.

First of all, multi-body models differ in the kind of coordinates used for their formulation. In particular, *independent* and *dependent* coordinates can be distinguished, the latter can be in turn divided into *relative*, *reference point* and *natural* coordinates. The choice of either type of coordinates can lead to significant differences in terms of efficiency and ease of implementation of the model.
An independent coordinate model of a mechanical system has a number of equations equal to the one of degrees of freedom, whereas the use of dependent coordinates leads to a larger model. Indeed, the use of dependent coordinates requires to add into the model some constraint equations expressing coordinate redundancy. It is apparent that size difference between the two models is equal to the number of constraint equations.
It is worth noticing that there is not a unique criterion for choosing independent or dependent coordinates, some authors however draw attention to the fact that the choice of independent coordinates does not allow the unambiguous determination of the position of all the system bodies [13]. This makes the method not general at all, so dependent coordinates are typically preferred, in spite of the model dimension increment.
Such considerations have led to the adoption of dependent coordi-

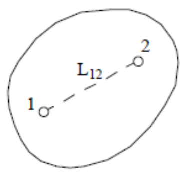

Figure 1: Rigid body defined by two points.

nates in buggy model formulation, thoroughly described in the following sections.

2.1.2 *Modelling using natural coordinates*

As it was told, modelling of a mechanical system using natural coordinates leads to an equation set whose dimension is higher than the one obtained modelling the system using independent coordinates. The size difference is equal to the one between the number of natural coordinates and the mechanism degrees of freedom. Added equations are algebraic and they can be divided into two kinds:

A. Equations which express the rigidity of a solid element, also called *rigid body constraints*;

B. Equations which force the relative movement between two joined elements to follow the one allowed by a kinematic pair, also called *joint constraints*.

These equations impose some constraints over the Cartesian coordinates of some points and unit vectors – sometimes the latter ones appear in spatial mechanisms – defined over the rigid bodies.

For planar mechanisms, considering for example the element of Fig. 1, rigid body constraints express the fact that the distance between the two points never changes. The corresponding equation is:

$$(x_2 - x_2)^2 + (y_2 - y_2)^2 - L_{12}^2 = 0 \tag{1}$$

In spatial mechanism case, considering the same element, equation (1) is enriched with a term along z axis:

$$(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2 - L_{12}^2 = 0 \tag{2}$$

If unit vectors are added, two kinds of equations are considered: the former one expresses the fact that vector magnitude never changes, i. e. it is always equal to one. Considering the rigid body of Fig. 2, the equation is:

$$u_x^2 + u_y^2 + u_z^2 - 1 = 0 \tag{3}$$

The second equation, through a dot product, expresses the fact that the value of the angle made by unit vector direction and the segment

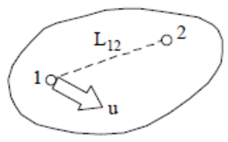

Figure 2: Rigid body defined by three points and a unit vector.

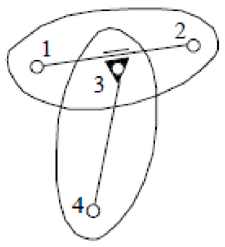

Figure 3: Prismatic joint in a planar mechanism.

passing through the two points never changes. Such an equation can be written in the following way:

$$(x_2 - x_1)u_x + (y_2 - y_1)u_y + (z_2 - z_1)u_z - c = 0 \tag{4}$$

Joint constraints express, through dot and cross products, the kind of movement allowed by them. For example, given the prismatic pair of Fig. 3, constraint equations are:

$$\begin{aligned}(x_2 - x_1)(y_3 - y_1) + (y_2 - y_1)(x_3 - x_1) = 0\\(x_2 - x_1)(x_4 - x_3) + (y_2 - y_1)(y_4 - y_3) - c = 0\end{aligned} \tag{5}$$

The former expresses, forcing the cross product to be equal to zero, the fact that points 3, 1 and 2 are always aligned. The latter forces the dot product to a constant value, with the aim of avoiding relative rotations between the two elements. In spatial mechanism case – as it is showed in Fig. 4 – the latter constraint can be expressed including one or more unit vectors into the dot product:

$$v_{1_x}v_{2_x} + v_{1_y}v_{2_y} + v_{1_z}v_{2_z} - c = 0 \tag{6}$$

2.1.3 *Kinematic analysis*

Kinematic analysis of a mechanical system consist of *position*, *velocity* and *acceleration* problems.
Position analysis, starting from the knowledge of the position of the degrees of freedom, aims at determining the position of each element of a mechanism. It can be solved, considering natural coordinate multi-body models, using numerical methods, like, for example, *Newton-Raphson* iterative one. This because constraint equations are

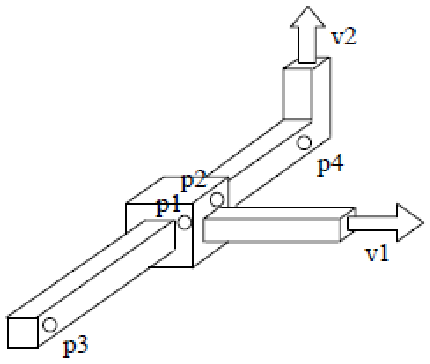

Figure 4: Prismatic joint in a spatial mechanism.

non linear: a closed-form solution which is also general and that can be found automatically does not exist.
Considering a Cartesian coordinate vector of positions **q** and a constraint equation vector ˘(**q**), system model can be written as:

$$\boldsymbol{\Phi}(\mathbf{q}) = \mathbf{0} \tag{7}$$

Using Taylor's series the model in (7) can be linearised in the neighbourhood of a position $\mathbf{q}_o$, leading to:

$$\boldsymbol{\Phi}(\mathbf{q}) \cong \boldsymbol{\Phi}(\mathbf{q_0}) + \boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q_0})(\mathbf{q} - \mathbf{q}_o) = \mathbf{0} \tag{8}$$

$\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q})$ is called *Jacobian matrix* of the system, its entries are the partial derivatives with respect to coordinate vector **q** of the constraint equations. The number of rows of $\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q})$ is equal to the one of constraint equations, while the number of its columns is equal to the number of coordinates. Basically, the dimensions of the Jacobian matrix are equal, respectively, to model and **q** vector dimensions. It is worth noticing that the values of some of the coordinates collected in **q** is known, since they are mechanism degrees of freedom: deleting the rows and columns related to them a well-posed system of equations can be obtained, whose dimension is equal to constraint equation number.
Let k be the current iteration step, the following equation has to be solved:

$$\boldsymbol{\Phi}_{\mathbf{q}}(q_k)(\mathbf{q}_{k+1} - \mathbf{q}_k) = -\boldsymbol{\Phi}(q_k) \tag{9}$$

So a new solution $\mathbf{q}_{k+1}$ is obtained, probably more similar to the real one than $\mathbf{q}_k$ was. The algorithm stops to iterate when the difference between $\mathbf{q}_k$ and $\mathbf{q}_{k+1}$ is negligible, so that it can be concluded that the convergence has been reached.

Velocity analysis leads to the determination of the speed of each element of the mechanism. Calculating the time derivative of (7), the following equation set is obtained:

$$\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{0} \tag{10}$$

When velocity problem is being solved, the position of all mechanism elements is already known. The Jacobian matrix is known too, so the equation system in (10) is linear. Moving the columns which are related to the degrees of freedom to the right side of the equation – their velocities have to be known – a square linear equation system is obtained.

Acceleration problem in linear too, so it can be solved easily using automatic calculation methods. Calculating the time derivative of (10), the following equation set is obtained:

$$\mathbf{\Phi_q}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{\Phi}}_\mathbf{q}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{0} \tag{11}$$

The solution method is equal to the one adopted for velocity problem, since the only terms which are not known make a square linear equation set whose dimension is equal to the number of constraint equations.

2.1.4 *Dynamic analysis*

Dynamic analysis of a mechanical system usually follows kinematic one. Its aim is to determine the forces which are acting on the mechanism. The most widespread approach to formulate dynamical model, when natural coordinates are used, is the Lagrange's one, following which the motion equations can be written as:

$$\frac{d}{dt}\frac{\partial \mathbf{T}}{\partial \dot{\mathbf{q}}} - \frac{\partial \mathbf{T}}{\partial \mathbf{q}} + \mathbf{\Phi_q}^T\boldsymbol{\lambda} = \mathbf{Q} \tag{12}$$

In (12) **T** is the kinetic energy of the system, **Q** is the vector of the generalized forces acting on it, the third term of the sum represents the forces due to the constraints between dependent variables **q**[1].
Let **M** be the mass matrix of the system, the kinetic energy can be written as follows:

$$\mathbf{T} = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}\dot{\mathbf{q}} \tag{13}$$

The result is that (12) can be written in a lighter and more intuitive form. Motion equation number is equal to the one of system independent variables, while the number of Lagrange multipliers is equal to the number of constraint equations. They still have to be calculated. Substituting (13) in (12), and taking advantage of the constraint equations defined in (7) the full equation set can be written as:

$$\begin{aligned} \mathbf{M}\ddot{\mathbf{q}} + \mathbf{\Phi_q}^T\boldsymbol{\lambda} &= \mathbf{Q} \\ \mathbf{\Phi} &= \mathbf{0} \end{aligned} \tag{14}$$

Equation (14) is made up of a set of differential equations and a set of algebraic equations. Therefore, motion equations in (14) result in

1 The term λ is called *Lagrange multiplier*.

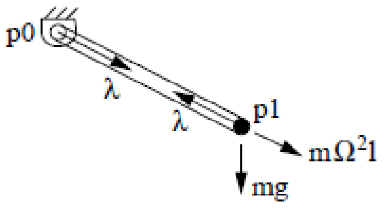

Figure 5: Physical meaning of Lagrange multipliers.

a set of *differential-algebraic equations* (DAE). However, most of the integration algorithms work with *ordinary differential equations* (ODE): a way to convert (14) in ODE is needed.
Several dynamic formulations for solving a problem like the one in (14) can be found in literature [13]. The one commonly known as *Augmented Lagrange* will be presented as an example. Starting from the first equation in that set, Lagrange multipliers can be expressed as:

$$\boldsymbol{\lambda} = \alpha(\ddot{\boldsymbol{\Phi}} + 2\xi\omega\dot{\boldsymbol{\Phi}} + \omega^2\boldsymbol{\Phi}) \tag{15}$$

where α is a coefficient called *penalty number*. Taking into account the physical meaning of the third term of the sum in (12) and having a glance at Fig. 5, α value has to be proportional to the violation of the constraint equations. The term in brackets in (15) can be explained as follows: each rigid body is replaced with a one degree of freedom spring-mass-damper system, so renouncing to its rigidity. This is replaced with a high stiffness: it is worth pointing out the importance of the right choice of penalty number value. Indeed, a high value leads to very high stiffness and facilitates constraint equation compliance, but it can lead to numeric problems. Opposite effects are obtained for small values of α[2].
In order to improve the accuracy of penalty number calculation, their definition can be slightly modified:

$$\boldsymbol{\lambda} = \boldsymbol{\lambda}^* + \alpha(\ddot{\boldsymbol{\Phi}} + 2\xi\omega\dot{\boldsymbol{\Phi}} + \omega^2\boldsymbol{\Phi}) \tag{16}$$

The result is an iterative algorithm in which the first term of the sum is the penalty number calculated at previous step. Substituting (16) into the first set of equations in (14) the following equation system is obtained:

$$\begin{aligned} (\mathbf{M} + \alpha\boldsymbol{\Phi_q}^T\boldsymbol{\Phi_q})\ddot{\mathbf{q}} = \mathbf{Q} - \boldsymbol{\Phi_q}^T[\boldsymbol{\lambda}_k + \alpha(\dot{\boldsymbol{\Phi}}_\mathbf{q}\dot{\mathbf{q}} + 2\xi\omega\dot{\boldsymbol{\Phi}} + \omega^2\boldsymbol{\Phi})] \\ \boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \alpha(\ddot{\boldsymbol{\Phi}} + 2\xi\omega\dot{\boldsymbol{\Phi}} + \omega^2\boldsymbol{\Phi}) \end{aligned} \tag{17}$$

where k is current iteration step.
Starting from an initial value of Lagrange multipliers, it can be used for acceleration calculation according to the first equation in (17), obtaining more accurate values which can be substituted into the second set of equations. The algorithm iterates until it converges.

2 Usually, heuristically, a right value for α can be chosen among 10^6 e 10^9.

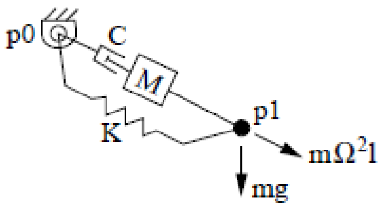

Figure 6: Physical meaning of penalty numbers.

2.1.5 *Integrators*

In multi-body dynamics, in addition to the formulations above, integration algorithms are very important too. They are characterized and classified according to three criteria:

A. Efficiency;

B. Stability;

C. Accuracy.

The first one does not need to be explained, the second one expresses the capability of the algorithm of converging for any value of the integration step width. Usually an integrator is said to be convergent if, at any time-step, it succeeds in calculating a finite solution, even though it is not the correct one. The third criterion is related to the correctness of the solution, distinguishing between *local* and *global* error, respectively committed over a single step or over the entire integration period. Integration algorithms can also be divided in:

A. *Single-step* integrators;

B. *Multiple-step* integrators.

The former ones compute the solution at $k+1$ step using only the values determined at k step, the latter ones use also the solutions calculated in the previous integration steps. Single-step integrators are used, usually, for complex integrations, with high-frequency solutions or discontinuities, whereas multiple-step integrators are more efficient in simple problems.
Another division can be made, distinguishing between:

A. *Fixed-step* integrators;

B. *Variable-step* integrators.

The former ones do not need any explication, the latter automatically vary integration step width depending on the complexity of the integration. When facing a critic point they make the step narrower, and vice versa.The former ones usually need less calculating capacity, and it seems to be preferable. Anyway, in presence of critic points

in the integrating function, the need for making the integration step narrower can appear, in order to avoid instability problems.
The last division of integration algorithms can be obtained highlighting the difference between:

A. *Explicit* integrators;

B. *Implicit* integrators.

The former ones calculate the solution at $k+1$ step as explicit function of the values determined at k step and of their derivatives. The latter ones also require the values of the solution at the current step, needing to solve a non-linear problem.
To solve the non-linear system which derives from the choice of implicit integrator, *fixed-point* techniques or *Newton-Raphson* algorithms can be used. They both utilize an explicit integrator, called *predictor*, in order to obtain an approximate solution. Then they perform the iterative calculation, called *corrector*, to converge to the solution they were looking for. The fixed point techniques use the implicit integrator as corrector, Newton-Raphson algorithms use tangent matrix algorithm for the same task.

An example of commonly used integrator is the *trapezoidal rule*. It is an implicit, single-step integration algorithm which is, moreover, unconditionally stable, so it works well with complex integrations.
It starts proceeding with the calculation of an approximate solution performed by the predictor:

$$\begin{aligned} \mathbf{q}_{k+1} &= \mathbf{q}_k + \Delta t \dot{\mathbf{q}}_k + \frac{\Delta t^2}{2}\ddot{\mathbf{q}}_k \\ \dot{\mathbf{q}}_{k+1} &= \dot{\mathbf{q}}_k + \Delta t \ddot{\mathbf{q}}_k \end{aligned} \tag{18}$$

The solution found is the starting point for the iterative process performed by the corrector, which tries to converge to the exact solution:

$$\begin{aligned} \mathbf{q}_{k+1} &= \mathbf{q}_k + \frac{\Delta t^2}{2}(\dot{\mathbf{q}}_k + \dot{\mathbf{q}}_{k+1}) \\ \dot{\mathbf{q}}_{k+1} &= \dot{\mathbf{q}}_k + \frac{\Delta t^2}{2}(\ddot{\mathbf{q}}_k + \ddot{\mathbf{q}}_{k+1}) \end{aligned} \tag{19}$$

2.2 EXAMPLE MODEL: THE FOUR BAR LINKAGE MECHANISM

In the following sections a brief description of a simple mechanism model in natural coordinates will be presented for sake of clarity, since vehicle one is quite complex to understand without being familiar with this kind of notation.

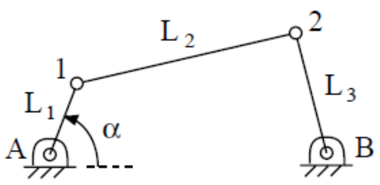

Figure 7: The four bar linkage mechanism.

2.2.1 *Kinematic model*

The four bar linkage mechanism, sketched in Fig. 7, consists of one fixed link and three moving ones. The former is called *frame*, the moving links connected to the frame are named *cranks* and finally the one between them is named *rod*. Such a mechanism has one degree of freedom, the use is to consider as actuated one of the cranks, for example by a motor. In this way, the angular position af the crank – denoted with α – is known.
Modelling using natural coordinates requires to write a set of constraint equations. The chosen coordinates are:

A. x_1, y_1

B. x_2, y_2

C. α

Since constraint equation set size is equal to the difference between the number of coordinates and the one of degrees of freedom, four equations are to be written. Three of them are rigid body constraints:

$$\begin{aligned}(x_1 - x_A)^2 + (y_1 - y_A)^2 - L_1^2 = 0\\(x_2 - x_1)^2 + (y_2 - y_1)^2 - L_2^2 = 0\\(x_B - x_2)^2 + (y_B - y_2)^2 - L_3^2 = 0\end{aligned} \tag{20}$$

The fourth expresses the relationship between α angle value and the length of the projection of the actuated crank. This equation can be written in two ways:

$$\begin{aligned}(x_1 - x_A) - L_1 \cos(\alpha) = 0\\(y_1 - y_A) - L_1 \sin(\alpha) = 0\end{aligned} \tag{21}$$

It is worth noticing that none of equations in (21) is always valid: the sine equation is not valid for α values in a neighbourhood of 90°, analogously cosine equation is not valid for α values around 0°. It can be explained by differentiating with respect to time the equations in (21): taking for example the former in (21), a small angle increment $\delta\alpha$ should produce small variations of other coordinate values:

$$(\delta x_1 - x_A) + L_1 \delta\alpha \sin(\alpha) = 0 \tag{22}$$

It can be noted that a small increment of α cannot induce variation in point 1 x coordinate, so this equation is not valid for $\alpha \cong 0°$. This is why the fourth equation of the constraint set must be chosen between the former and the latter of (21), depending on α value.

2.2.2 *Mass and force evaluation*

In section 2.1.4 the formulation of dynamical models using natural coordinates has been discussed in a general way. However, the mass matrix **M** and the force vector **Q** were introduced without any kind of explanation. In this section come clarifications on the formulation of these terms are provided for the specific case analysed.
Mass matrix is evaluated moving from energetic considerations: given for example the rod of Fig. 7, the position of point 2 with respect to a generic reference frame can be expressed as:

$$\mathbf{r} = \mathbf{r_1} + \mathbf{A}\bar{\mathbf{r}} \tag{23}$$

where **A** is the rotation matrix which expresses crank local axes with respect to global reference frame. It can be written as follows:

$$\mathbf{A} = \frac{1}{L_2}\begin{bmatrix} x_2 - x_1 & y_1 - y_2 \\ y_2 - y_1 & x_2 - x_1 \end{bmatrix} \tag{24}$$

Evaluating the time derivative of (23) and the kinetic energy of the system, the following equation can be obtained:

$$\mathbf{T} = \frac{1}{2}\int \dot{\mathbf{r}}_1^T \dot{\mathbf{r}}_1 dm + \frac{1}{2}\int \dot{\mathbf{r}}_1^T \dot{\mathbf{A}}\bar{\mathbf{r}} dm + \frac{1}{2}\int \bar{\mathbf{r}}^T \dot{\mathbf{A}}^T \dot{\mathbf{r}}_1 dm + \frac{1}{2}\int \bar{\mathbf{r}}^T \dot{\mathbf{A}}^T \dot{\mathbf{A}}\bar{\mathbf{r}} dm \tag{25}$$

The equation in (25) can be integrated, resulting in:

$$\mathbf{T} = \frac{1}{2}m\dot{\mathbf{r}}_1^T \dot{\mathbf{r}}_1 + m\dot{\mathbf{r}}_1^T \dot{\mathbf{A}}\bar{\mathbf{r}}_G + \frac{I_2}{2L_2}[(\dot{x}_2 - \dot{x}_1)^2 + (\dot{y}_2 - \dot{y}_1)^2] \tag{26}$$

where $\mathbf{r_G}$ is the local position vector of rod centre of gravity and I_2 is its moment of inertia with respect to point 1, in which the origin of local reference frame has been placed. Kinetic energy of the system can be written also as:

$$\mathbf{T} = \frac{1}{2}\dot{\mathbf{x}}\begin{bmatrix} m - 2m\frac{\bar{x}_G}{L} + \frac{I_2}{L_2^2} & 0 & m\frac{\bar{x}_G}{L} - \frac{I_2}{L_2^2} & -m\frac{\bar{y}_G}{L} \\ 0 & m - 2m\frac{\bar{x}_G}{L} + \frac{I_2}{L_2^2} & m\frac{\bar{y}_G}{L} & m\frac{\bar{x}_G}{L} - \frac{I_2}{L_2^2} \\ m\frac{\bar{x}_G}{L} - \frac{I_2}{L_2^2} & m\frac{\bar{y}_G}{L} & \frac{I_2}{L_2^2} & 0 \\ -m\frac{\bar{y}_G}{L} & m\frac{\bar{x}_G}{L} - \frac{I_2}{L_2^2} & 0 & \frac{I_2}{L_2^2} \end{bmatrix}\dot{\mathbf{x}}^T \tag{27}$$

where $\dot{\mathbf{x}}$ is the vector of natural coordinate velocity. It is easy to notice that the term in the middle is the mass matrix.

It is worth pointing out that this matrix has constant values, so the force vector **Q** will not contain inertial components of the forces – *centrifugal* and *Coriolis* one – which depends on velocity.

In order to calculate the generalized force vector **Q**, the velocity of a generic point, P, the rod is introduced. Such a velocity can be expressed, with respect to a generic reference frame, as follows:

$$\dot{\mathbf{r}}_{\mathbf{P}} = \mathbf{C}_{\mathbf{P}}\dot{\mathbf{q}} \tag{28}$$

where $\mathbf{C}_{\mathbf{P}}$ is a matrix which conveys point P position in the element base. In order to determine the force which provokes this velocity, its *virtual power* has to be evaluated. Two force systems are said to be dynamically equivalent if they produce the same virtual power:

$$\dot{\mathbf{W}} = \mathbf{F}^T\dot{\mathbf{r}}_{\mathbf{P}} = \mathbf{F}^T\mathbf{C}_{\mathbf{P}}\dot{\mathbf{q}} = \mathbf{Q}^T\dot{\mathbf{q}} \tag{29}$$

In this case a plane mechanism is being considered, so that (23) can be rewritten as:

$$\mathbf{r}_{\mathbf{P}} = \mathbf{r}_1 + a\mathbf{u} + b\mathbf{v} \tag{30}$$

where **u** and **v** are two unit vectors:

$$\begin{aligned} \mathbf{u} &= \frac{1}{L_2}\begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix} \\ \mathbf{v} &= \frac{1}{L_2}\begin{bmatrix} y_1 - y_2 \\ x_2 - x_1 \end{bmatrix} \end{aligned} \tag{31}$$

Rewriting the equation in (30) the following relation can be obtained:

$$\mathbf{r}_{\mathbf{P}} = \frac{1}{L_2}\begin{bmatrix} L_2 - a & b & a & -b \\ -b & L_2 - a & b & a \end{bmatrix}\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \end{bmatrix} = \mathbf{C}_{\mathbf{P}}\mathbf{q} \tag{32}$$

from which the value of $\mathbf{C}_{\mathbf{P}}$ matrix is obvious.
The mass matrix and force vector obtained from this calculation must be assembled. Every term in these is related to a coordinate: they simply have to be put in the correct position of the complete matrix, paying heed to sum the terms which are related to the same variable.

2.2.3 *Multi-body simulation*

The multi-body simulation starts with a kinematic analysis: position and velocity initial conditions of the degrees of freedom are known, so positions and velocities of all the coordinates can be evaluated.

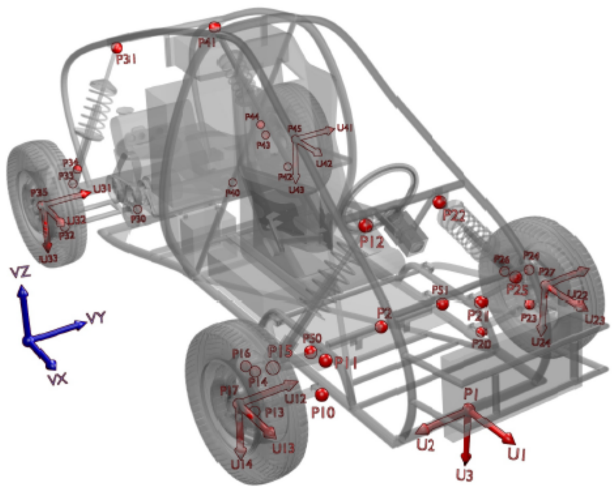

Figure 8: Buggy model, highlighting natural coordinates.

At any time-step, a dynamic analysis is performed: firstly position, velocity and acceleration values are pre-calculated, following the equations in (18). Then the non-linear problem has to be solved: using the equations in (19) accurate kinematic variable values can be obtained. Once that the convergence has been reached, the dynamic analysis ends: all the coordinate values of the mechanism at the current time-step are known.
This calculation provides the initial conditions for the next integration step: in this way mechanism dynamics can be determined for the entire simulation period[3].

2.3 VEHICLE MODEL

2.3.1 *Multi-body modelling*

As it was told before, the multi-body model of the buggy has been essentially built using natural coordinates, even though some relative ones have been added for convenience. The buggy is considered to be made of eighteen rigid bodies, all defined by points and unit vectors, so the corresponding leading model has 168 coordinates [26].

The chassis, shown in Fig. 8, has been modelled using twelve basic points and three unit vectors. For the purpose of this work, p_1 point and the three unit vectors $\mathbf{u_1}$, $\mathbf{u_2}$, $\mathbf{u_3}$ have very great importance. Indeed, they have been used to observe and control the six degrees of freedom of buggy chassis, since its position and orientation can be completely defined by them.
In particular, buggy position in the three-dimensional space can be determined looking at the three Cartesian coordinates of p_1 point.

3 MATLAB code of four bar linkage multi-body simulation can be found in Appendix A

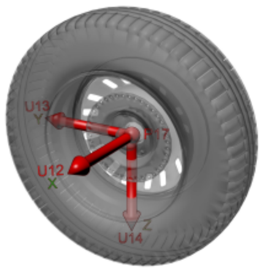

Figure 9: Front left wheel model, highlighting natural coordinates.

Regarding to the orientation, taking into account that buggy transformation matrix is made of the three unit vectors $\mathbf{u_1}$, $\mathbf{u_2}$, $\mathbf{u_3}$, the following components of them can be chosen in order to make the chassis rotate:

A. y component of $\mathbf{u_1}$ vector

B. z component of $\mathbf{u_1}$ vector

C. z component of $\mathbf{u_2}$ vector

It is worth noticing that *it is not a linearisation*, even if for small values of the three angles there is a similarity with Tait-Bryan angles. Natural coordinate models simply use points and unit vectors to express rigid body movements, so that the three orientation angles we are used to think at – yaw, pitch and roll – *do not exist*: their task is performed through the modification of Cartesian coordinates.
Since the chassis variables are 45 – 36 coordinates of the points and 9 components of the unit vectors – while its degrees of freedom are six, 39 constraint equations have to be written. In particular, rigid body constraints are expressed imposing respectively orthogonality and unit magnitude of the three vectors:

$$\begin{aligned} \mathbf{u_i}\mathbf{u_j} &= 0 \\ \mathbf{u_i}\mathbf{u_i} - 1 &= 0 \end{aligned} \tag{33}$$

The rest of the constraint equations define the other points as linear combination of the three unit vectors:

$$\mathbf{r_m} - a\mathbf{u_i} - b\mathbf{u_j} - c\mathbf{u_k} = \mathbf{0} \tag{34}$$

where $\mathbf{r_m}$ is the distance of p_m from p_1.

Each wheel – in Fig. 9 the front right one is shown – is defined by one point and three unit vectors. Constraint equations for wheels are the ones of rigid body, expressed as follows:

$$\begin{aligned} \mathbf{u_i}\mathbf{u_i} - 1 &= 0 \\ \mathbf{u_i}\mathbf{u_j} - \cos(\phi_{ij}) &= 0 \end{aligned} \tag{35}$$

Actually, some relative coordinates have been added to natural ones, in order to easily describe the movement allowed by kinematic joints.

For example, the rotation of the wheels is described through the value of an angle θ: so there are four angular variables. These angles are defined with respect to:

A. The projection of the two points of the knuckle onto the wheel plane and a vector contained in the latter, for front wheels;

B. The projection of a vector of the chassis onto the wheel plane and a vector contained in the latter, for rear wheels;

Constraint equations for front wheels are:

$$\begin{aligned} \mathbf{u_i r_k} - (\mathbf{u_i u_j})(\mathbf{r_k u_j}) - L\cos(\theta_n) = 0 \\ \mathbf{u_i} \times \mathbf{r_k} - (\mathbf{u_i u_j})\mathbf{u_j} \times \mathbf{r_k} - (\mathbf{r_k u_j})\mathbf{u_j} \times \mathbf{u_j} - \mathbf{u_j} L\sin(\theta_n) = \mathbf{0} \end{aligned} \tag{36}$$

The same equations for rear wheels are:

$$\begin{aligned} \mathbf{u_i u_j} - (\mathbf{u_i u_k})(\mathbf{u_j u_k}) - L\cos(\theta_n) = 0 \\ \mathbf{u_i} \times \mathbf{u_j} - (\mathbf{u_i u_j})\mathbf{u_k} \times \mathbf{u_j} - (\mathbf{u_j u_k})\mathbf{u_k} \times \mathbf{u_k} - \mathbf{u_k} L\sin(\theta_n) = \mathbf{0} \end{aligned} \tag{37}$$

The stroke of the dampers has been defined, like the steering mechanism rack translation, using relative coordinates. In particular, these are longitudinal distances $\mathbf{s}$ and lead to constraint equation as:

$$\mathbf{s_i} = |\mathbf{r_{jk}}| \tag{38}$$

where $\mathbf{r_{jk}}$ is the distance between p_j and p_k.

2.3.2 *Force modelling*

Several kinds of force models are included, which are:

A. Tyre forces;

B. Gravity forces;

C. Suspension forces;

D. Driving and braking torques of rear wheels;

E. Braking torques of front wheels.

The first ones calculate vertical and lateral forces generated by the interaction between tyres and the road profile, *which is supposed to be known*. The vertical behaviour is the same of a one degree of freedom spring-mass-damper system, the lateral behaviour is a bit more complex and can be represented as proportional to the slip or through other – more complicated – models.
Gravitational forces act on the centre of gravity of each rigid body, in the direction of $\mathbf{v_z}$ vector – showed in Fig. 8 – with a value of the gravity acceleration of $g = 9.81\ \frac{m}{s^2}$.

2.3.3 *Dynamic formulation*

The dynamic formulation used in buggy model is an improved version of the *augmented Lagrangian* method in (17), in particular it is called *index-3 augmented Lagrangian formulation with mass-damping-stiffness-orthogonal projections in velocities and accelerations* [7, 9]. It, in turn, is a modified version of the *index-3 augmented Lagrangian formulation with mass-orthogonal-projections* [5], from which the equations of motion are derived:

$$\begin{aligned} \mathbf{M}\ddot{\mathbf{q}} + \mathbf{\Phi_q}^T \alpha \mathbf{\Phi} + \mathbf{\Phi_q}^T \boldsymbol{\lambda}^* &= \mathbf{Q} \\ \boldsymbol{\lambda}^*{}_k &= \boldsymbol{\lambda}^*{}_{k-1} + \alpha \mathbf{\Phi}_k \end{aligned} \tag{39}$$

The implicit single-step trapezoidal rule has been chosen as integration algorithm. This leads to:

$$\begin{aligned} \dot{\mathbf{q}}_{k+1} &= \frac{2}{\Delta t}\mathbf{q}_{k+1} - (\frac{2}{\Delta t}\mathbf{q}_k + \dot{\mathbf{q}}_k) \\ \ddot{\mathbf{q}}_{k+1} &= \frac{4}{\Delta t^2}\mathbf{q}_{k+1} - (\frac{4}{\Delta t^2}\mathbf{q}_k + \frac{4}{\Delta t}\dot{\mathbf{q}}_k + \ddot{\mathbf{q}}_k) \end{aligned} \tag{40}$$

Substituting the difference equations in (40) into (39) the non-linear system can be solved using Newton-Rapshon method:

$$\begin{aligned} [\frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}]_k \mathbf{\Delta q}_{k+1} &= -[f(\mathbf{q})]_k \\ [f(\mathbf{q})] &= \frac{4}{\Delta t^2}(\mathbf{M}\ddot{\mathbf{q}} + \mathbf{\Phi_q}^T \alpha \mathbf{\Phi} + \mathbf{\Phi_q}^T \boldsymbol{\lambda}^* - \mathbf{Q}) \end{aligned} \tag{41}$$

The position solution set $\mathbf{q}_{k+1}$ satisfies both the equations of motion in (39) and the constraint equations in (7). Yet velocity and acceleration solution sets do not satisfy the constraints in (10) and (11), since it has not been imposed in the solution process. Because of this, velocity and acceleration projections are performed after Newton-Raphson algorithm convergence:

$$\begin{aligned} [\mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}(\mathbf{\Phi_q}^T \alpha \mathbf{\Phi} + \mathbf{K})]\dot{\mathbf{q}} &= [\mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}\mathbf{K}]\dot{\mathbf{q}}^* + \frac{\Delta t^2}{4}\mathbf{\Phi_q}^T \alpha \mathbf{\Phi} \\ [\mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}(\mathbf{\Phi_q}^T \alpha \mathbf{\Phi} + \mathbf{K})]\ddot{\mathbf{q}} &= [\mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}\mathbf{K}]\ddot{\mathbf{q}}^* + \frac{\Delta t^2}{4}\mathbf{\Phi_q}^T \alpha (\dot{\mathbf{\Phi}}_\mathbf{q}\dot{\mathbf{q}} + \dot{\mathbf{\Phi}}) \end{aligned} \tag{42}$$

where $\mathbf{C}$, $\mathbf{K}$ are the contributions of damping and elastic forces of the system, respectively, $\dot{\mathbf{q}}^*$ is the solution obtained after Newton-Raphson convergence – idem for accelerations – and $\dot{\mathbf{q}}$ is the one after the projection.

3 THE OBSERVER

3.1 STATE ESTIMATION

3.1.1 *General concepts*

Before explaining state estimation concept, the term *state* should be defined. The state of a dynamical system is a set of variables whose values completely represent system internal conditions at a given instant of time [31]. More formally, the state of a dynamical system is defined as a set of variables which have the *separation property*. This means that they contain all the system history information which are needed in order to evaluate the evolution both of output variables and of state ones [12].
The knowledge of the values of system state variables is fundamental if an advanced control technique has to be implemented: in fact several optimal control strategies suppose the system state to be fully available. However, state variables often cannot be measured: sometimes they refer to system parts which are difficult to reach, sometimes a suitable sensor would be too much expensive. This is the reason for the growth of state estimation importance in several fields of engineering research.

A *state observer* is a dynamical system which performs the approximate calculation of state variables, using disposal data. The most easily understandable one is *Luenberger observer* [19]: by using a model of the real system, and by applying it the same inputs, such an observer computes the estimates of system variables output. Finally, through the feedback of the output errors, weighted by a gain matrix, it succeeds in determining the system model state. The main problem relies in the fact that both the system model and the input variables are treated in a deterministic way, i. e. uncertainties on model and noise on measurements are completely neglected, leading to significant limitations [20].
For this reason the *Kalman filter* is often used for the same task: it performs state estimation taking into account uncertainties and noise defining all the variables involved in the estimation like stochastic ones.

3.1.2 *Observability*

Regarding to state estimation, a key concept is *observability*, firstly introduced by R. E. Kalman. In fact, it is strictly related to the possibility

of determining system state, moving from the knowledge of its structure and outputs.
Formally speaking, a state which produces outputs which are undistinguishable from the ones produced by the null state is called *unobservable*. So the subspace containing it is called *non-observability subspace*. If the subspace contains only the null state, the system is said to be observable [12]. In a more intuitive way, a system with an initial state $\mathbf{x}(t_0)$ is said to be observable if such a state can be determined from its outputs in the time interval $[t_0, t]$.
Given a discrete-time linear-time-invariant system in state-space form[1], this property can be proved taking a look to *observability matrix*:

$$\mathbf{O} = \begin{bmatrix} \mathbf{H} \\ \mathbf{HF} \\ \mathbf{HF}^2 \\ \vdots \\ \mathbf{HF}^{n-1} \end{bmatrix} \tag{44}$$

where $\mathbf{F}$ is the system matrix, $\mathbf{H}$ is the output matrix, n is state dimension. In particular, if rank of the observability matrix is equal to n, the system is fully observable.
Several criteria for determining the observability of non-linear systems can be found in literature, in particular underlining the concepts of *local* and *weak* observability [15].

3.2 THE KALMAN FILTER

3.2.1 *Linear Kalman filter*

The Kalman filter is an algorithm which performs the estimation of the instantaneous state of a linear dynamic system, solving what is called *linear-quadratic problem* [14]. It is capable of combining the information deriving from the knowledge of system dynamics with the ones resulting from the measurements of noise-corrupted system output variables.
Although the Kalman filter is an alternative representation of the Wiener one, the credit of this formulation is usually given to R. E. Kalman, for connecting state estimation problem with state-space models [3, 17].
The Kalman filter works admitting neither the perfect knowledge of system dynamics, nor the capability of taking noise-free measurements of output variables. In particular, the gain is calculated – while

1 Let the system be expressed as:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{F}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k \end{aligned} \tag{43}$$

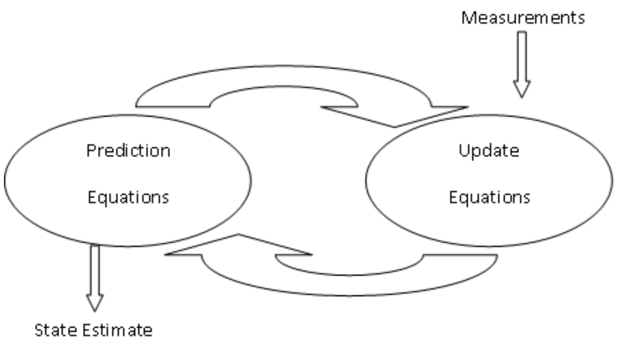

Figure 10: Kalman filter recursive process.

in Luenberger observer it is fixed and chosen by the designer – by mean of an algorithm, depending on how much the model and the measurements are being trusted.

It works recursively and it is relies of two steps:

A. *Prediction* or *propagation* step;

B. *Correction* or *updating* step.

In the former a *prediction* of state variable values and of their covariances, based on state-space system model, is performed. The estimation uses all the information until the current time-step, trying to predict what will happen at the next one. Given the state-space representation of a LTI discrete-time[2] dynamic system:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{F}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{J}\mathbf{u}_k \end{aligned} \tag{46}$$

The uncertainty over the model is expressed through the *plant noise matrix*, denoted here as $\mathbf{W}$, which has the same dimension of the state vector. Since it contains terms of uncertainty for each state component, it indicates how much they are distrusted.

The prediction-step Kalman filter equations are:

$$\begin{aligned} \mathbf{x}_{k|k-1} &= \mathbf{F}\mathbf{x}_{k-1} \\ \mathbf{P}_{k|k-1} &= \mathbf{F}\mathbf{P}_{k-1}\mathbf{F}^T + \mathbf{W} \end{aligned} \tag{47}$$

2 Given a LTI continuous-time system in state-space form, characterized by the $(\mathbf{F}, \mathbf{G}, \mathbf{H}, \mathbf{J})$ matrix set, it can always be converted into a discrete-time one. For example, the system in (46), could result from the discretisation of a continuous-time one using the formulas:

$$\begin{aligned} \mathbf{F} &= e^{\mathbf{A}T} \\ \mathbf{G} &= \int_0^T e^{\mathbf{A}s}\mathbf{B}ds \\ \mathbf{H} &= \mathbf{C} \\ \mathbf{J} &= \mathbf{D} \end{aligned} \tag{45}$$

where T is time-step value.

It is worth noticing that during this step *only the knowledge about system model* is used, taking into account the uncertainty on it through the inclusion of $\mathbf{W}$ matrix in updating the covariance.
The second step performs the correction of the predicted estimates, using measurement information. Measurement noise is expressed using the *measurement noise matrix*, denoted here as $\mathbf{N}$, which has a number of rows equal to measurement one, and a number of columns equal to state dimension one. Matrix $\mathbf{N}$ expresses how much each sensor measurement is distrusted.
A parameter, called *Kalman gain*, is computed:

$$\mathbf{L}_k = \mathbf{P}_{k-1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k-1}\mathbf{H}^T + \mathbf{N})^{-1} \tag{48}$$

The *estimation error*, also called *innovation*, is computed as follows:

$$\mathbf{e}_k = \mathbf{y}_k - \mathbf{H}\mathbf{x}_k \tag{49}$$

So the effective correction can be performed:

$$\begin{aligned} \mathbf{x}_{k|k} &= \mathbf{x}_{k|k-1} + \mathbf{L}_k\mathbf{e}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{L}_k\mathbf{H})\mathbf{P}_{k|k-1}(\mathbf{I} - \mathbf{L}_k\mathbf{H})^T + \mathbf{L}_k\mathbf{N}\mathbf{L}_k^T \end{aligned} \tag{50}$$

where $\mathbf{I}$ is the identity matrix[3].
It is worth pointing out that $\mathbf{P}$ matrix influences state correction through the Kalman gain. In fact, gain value is driven by uncertainties on state variables: this, combined with measurement noise, leads to determine *how much* the estimates need to be corrected.

The filter is a *recursive* algorithm, i. e. at any time-step it performs prediction and updating moving from the values determined at the previous one. Moreover, it should be reminded that it is a dynamic system too: so its poles determine its stability and its time constant. Regarding to the former, covariance prediction and updating equations could be combined into *discrete Riccati equation*:

$$\mathbf{P}_{k+1|k} = \mathbf{F}_k\mathbf{P}_{k|k}\mathbf{F}_k^T + \mathbf{F}_k\mathbf{P}_{k|k}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k}\mathbf{H}_k^T + \mathbf{N}_k)^{-1}\mathbf{H}_k\mathbf{P}_{k|k}\mathbf{F}_k^T + \mathbf{W}_k \tag{51}$$

It can be proved that a sufficient condition for covariance matrix pole stability is the *revealability* of $(\mathbf{F}, \mathbf{H})$ and the *stabilisability* of $(\mathbf{F}, \mathbf{Q}^{\frac{1}{2}})$.

3 There are three different expressions for the updating equation of the covariance matrix, the one presented is called *Joseph stabilized version* and it can be shown to be more stable and robust than the others, since it guarantees $\mathbf{P}$ matrix to be always symmetric and positive definite [31].

3.2.2 *The Extended Kalman filter*

The Extended Kalman filter is the most common and most used attempt to apply the Kalman filter to non-linear systems, described by the equations:

$$\begin{aligned} \mathbf{x}_{k+1} &= f(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{y}_k &= h(\mathbf{x}_k) \end{aligned} \tag{52}$$

The Extended Kalman filter simply linearises all the non-linear transformations and substitutes their Jacobians in order to compute covariance matrices and Kalman gain [16]. Linearisations are performed about the current mean and covariance:

$$\begin{aligned} \mathbf{F} &= \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}_0} \\ \mathbf{H} &= \frac{\partial h}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}_0} \end{aligned} \tag{53}$$

where $\mathbf{x}_0$ is the current working point. If the system does not have a fixed working point and it is time-varying these matrices have to be calculated at every time-step.
With respect to the linear version, this filter loses the optimality property: the main flaw of the EKF is that the distributions of the various random variables are no longer normal after undergoing their respective non-linear transformations. It is simply an *ad hoc* state estimator which approximates Bayes' rule of optimality by linearization [35].
The EKF has two implementation forms in terms of state estimation: the *total-state* and the *error-state* ones. In the former the object of the estimation is the total state of a system, in the latter – which will be presented in the next paragraph – observer state is made of the errors committed over system state variables.

3.2.3 *The error-state EKF*

The *error-state* EKF, also called *indirect* EKF, performs the estimation of the errors committed in the determination of the values of system state variables. That is, given a dynamical system with a couple of state variables (x_1, x_2), this kind of filter aims estimating the value of $(\delta x_1, \delta x_2)$.
The examples in literature [29, 28, 1, 32] refer to navigation problems, in particular to orientation estimation. As it can be noticed, dynamic modelling of the system in the state-transition matrix is not necessary: it is substituted by kinematic equation integrations. This leads to two main advantages:

A. The same observer can be used with other systems of the same kind, making it more general;

B. If not all the state variables are to be estimated, observer size can be reduced, making it smaller and faster. This is not possible with the total-state filter, since into the state-transition matrix there are information about system behaviour.

It will be seen, however, that some drawbacks are present: in particular the loss of part of system observability is probable.

3.3 ERROR-STATE EKF FOR A FOUR-BAR LINKAGE MECHANISM

Before working on the vehicle model, the approach proposed through this work has been tested on a very simple, one DOF mechanism: the four-bar linkage one. The same one which had been modelled in the previous chapter was used.

3.3.1 *The errorEKF*

The error-state EKF used in this work is a slightly modified version of the one designed in the papers quoted above: in particular, it was thought up to be suitable for multi-body systems and named *error*EKF [30]. It has been chosen, among various filters, because it results the best trade-off between accuracy and efficiency. In particular, although the *unscented Kalman filter* offers the best performances in terms of accuracy, typically it requires a higher computational cost than the EKF.
The errorEKF works integrating multi-body model variables, in fact its state-transition matrix is an integrator:

$$\mathbf{e}_{k+1} = \begin{bmatrix} \mathbf{e}_q \\ \mathbf{e}_{\dot{q}} \end{bmatrix}_{k+1} = \begin{bmatrix} \mathbf{I} & \Delta t \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{e}_q \\ \mathbf{e}_{\dot{q}} \end{bmatrix}_{k} \qquad (54)$$

This accords to the first of the advantages of the error-state EKF underlined in the last paragraph. It is worth noticing that the filter is input-free: it works in free response integrating the error vector – *which is the object of the estimation* – at the previous time-step. These errors are the difference between the values of real mechanism independent coordinates and multi-body model ones. Once they have been estimated, the errors in all model natural coordinates can be determined through kinematic analysis.
After that, since the errors committed on the multi-body systems state variables are known, a model correction is possible. So, the initial conditions for the next multi-body integration step will be the results of the previous time-step, corrected by filter estimates. This allows forcing to zero the predicted error vector: in fact, since the multi-body model has been corrected, the errors at the beginning of the current time-step are zero, as well as their predicted values, according to (54).

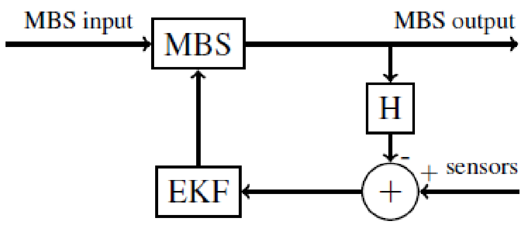

Figure 11: ErrorEKF simplified flow diagram.

The output matrix is calculated like in (53): it is, actually, the only non-linear part of the filter, since the Jacobian of $f(\mathbf{x}_k)$ function is replaced by the integrator. This matrix is made of the partial derivatives of sensors functions with respect to model variables.

3.3.2 *ErrorEKF stability*

Usually, before designing an observer, system observability analysis should be carried out, since observability is a precondition to perform state estimation. It provides also a criterion for Kalman filter stability: if the couple $(\mathbf{F}, \mathbf{H})$ is observable – and $(\mathbf{F}, \mathbf{Q}^{\frac{1}{2}})$ is reachable – surely $\mathbf{P}$ matrix has stable poles, so Riccati equation will have convergent solutions.
In this case, however, the $\mathbf{F}$ matrix which appears in (51) is not the Jacobian of $f(\mathbf{x}_k)$ function. So, simply being an integrator, it contains less information: this could result in a filter covariance divergence, even though the system is observable. This is easy to realize, thinking of the physical problem: in such a matrix there is not the information about system dynamics, for example how the movement of an element influences other, whereas in the total-state Kalman filter it was contained into the dynamic model.

Anyway, since the Kalman gain is still calculated solving Riccati equation, the same test of observability can be used for studying the stability of the filter, taking care of substituting system state-transition matrix with the integrator. This might result in a loss of observability. Moreover, it is worth pointing out that in this case revealability and observability correspond, since the integrator matrix has all its eigenvalues on the unit circle.
Therefore, in order to stabilize the filter, a higher number of measurements are required, if the system has more than one degree of freedom. This will not happen in the four bar linkage mechanism, since if the position of one crank is known, the dynamics of the entire mechanism can be estimated. It will happen, however, in the buggy, as it will be discussed in the next chapter.

3.3.3 *Mechanism observer*

Considering the four bar linkage mechanism illustrated in the previous chapter, the observer was firstly tested on it. The model has one degree of freedom, so that the errors to be estimated are two: position and velocity of the independent variables:

$$\mathbf{x} = \begin{bmatrix} z \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \alpha \\ \dot{\alpha} \end{bmatrix} \tag{55}$$

Because of that, the state-transition matrix is the same as in (54).
The mechanism is assumed to have an encoder on the crank: according to the state-space formulation, the output matrix results:

$$\mathbf{H} = \frac{\partial h}{\partial \mathbf{x}} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{56}$$

As it was told before, the filter works recursively through prediction and updating steps. In the first step the error vector can be, because of the multi-body model correction, forced to zero. This leads to the same equations as in (47), yet the prediction of the error is zero:

$$\begin{aligned} \mathbf{e}_{k|k-1} &= \mathbf{0} \\ \mathbf{P}_{k|k-1} &= \mathbf{F}\mathbf{P}_{k-1}\mathbf{F}^T + \mathbf{W} \end{aligned} \tag{57}$$

After prediction step, multi-body model integration is performed. This provides some *virtual measurements*: the sensors which are present on the mechanism are modelled too, so producing some outputs. The Kalman gain is computed as in (48), and then the innovation is evaluated as the difference between a set of system measurements $\mathbf{m}$ and the virtual ones $\mathbf{m_v}$.
Innovation vector is needed to evaluate the updating step in (50). After that, multi-body model correction can be performed. In particular, starting from the knowledge of the error vector of the updated independent variables, the errors on all the natural coordinates can be computed by solving the kinematic velocity problem:

$$\begin{aligned} \mathbf{\Phi_q}(\mathbf{q})\mathbf{e_{\dot{q}}} &= \mathbf{0} \\ \mathbf{\Phi_q}(\mathbf{q})\mathbf{e_q} &= \mathbf{0} \end{aligned} \tag{58}$$

Once the errors on all natural coordinates have been determined, multi-body model correction can be performed by adding to each coordinate the correspondent error.
It is worth noticing that position error values are calculated in an approximate way, solving the velocity problem. This because, in the case of infinitesimal position displacement, the following equation holds:

$$\frac{\partial \dot{\mathbf{q}}}{\partial \dot{\mathbf{z}}} = \frac{\partial \mathbf{q}}{\partial \mathbf{z}} = \mathbf{R} \tag{59}$$

The first term in (59) is the definition of the matrix which relates independent coordinate velocities to natural coordinate ones and it is called R matrix. If the hypothesis is satisfied, equation (59) holds and, as a consequence, velocity problem resolution can provide quite accurate position values. The main advantage is that the solution of velocity problem does not require any iterative calculation. Thus, it is expected to be more rapid, and more suitable to a real-time oriented problem.

It is worth pointing out that, with the sensor specified, the only one non-linearity of the filter is no more present: in fact, the first term of the output matrix means that the first state variable is *exactly* known. As a matter of fact, in this case the filter is linear. This means that, if it is working optimally, a white innovation can be expected: evaluating it the operating performances of the observer can be discussed and improved, through parameters tuning.
Among the methods in literature [20] the *cumulative periodogram* one has been used. It works evaluating innovation spectral density and verifying that it is constant in all the frequency domain [4]. It follows that, called σ_e the error variance and S_e its spectrum, its integral with respect to the frequency is a straight line:

$$S_e(e^{j\theta}) = \sigma_e^2 \quad \Rightarrow \quad I_e(\theta) = \int_0^\theta S_e(e^{jh})dh = \theta\sigma_e^2 \tag{60}$$

The *periodogram* is defined as the squared absolute value of innovation Fourier transform. That is, it is used in order to estimate error spectrum:

$$\begin{aligned} \hat{S}_e(\theta_k) &= |E(\theta_k)|^2 \\ E(\theta_k) &= \frac{1}{\sqrt{N}} \sum_{h=1}^{N} e(h)e^{-j\theta_k h} \end{aligned} \tag{61}$$

If innovation spectrum is white, estimated periodogram integral with respect to the frequency will be a straight line, too:

$$\hat{I}_e(\theta_{\bar{k}}) = \int_0^{\theta_{\bar{k}}} \hat{S}_e(h)dh \cong \sum_{h=0}^{\bar{k}} \hat{S}_e(\theta_k)\frac{2\pi}{N} \tag{62}$$

An example of great result, in terms of innovation whiteness, is shown in Fig. 12.
Another aspect, which is related to measurements, can be underlined: since α angle is measured directly, the couple $(\mathbf{F}, \mathbf{H})$ is observable. In fact, although state-transition matrix does not contain dynamic information, observability one rank is equal to system size:

$$\text{rank}(\mathbf{O}) = \text{rank}\begin{bmatrix} 1 & 0 \\ 1 & \Delta t \end{bmatrix} = 2 \tag{63}$$

So, the filter is supposed to be stable.

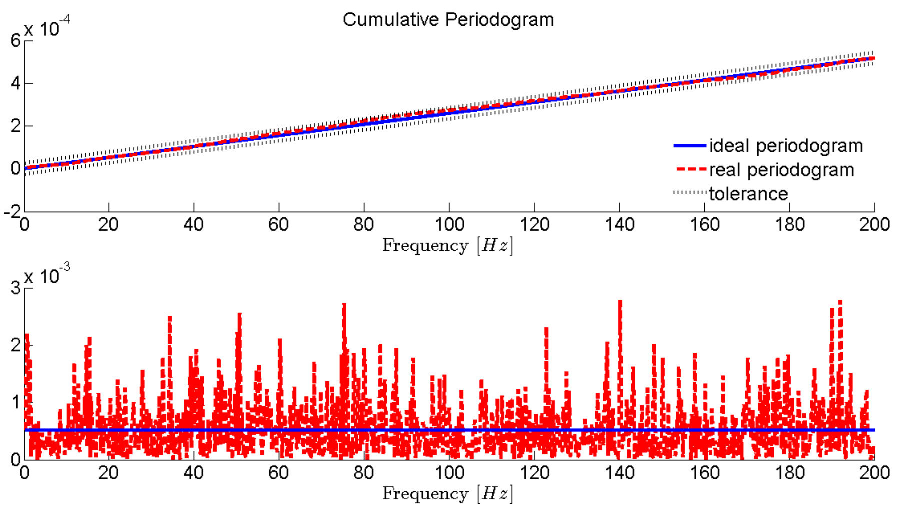

Figure 12: Cumulative periodogram test.

3.3.4 *Simulation*

The first simulations of the observer has been performed on this simple mechanism, since the results are easier to understand than vehicle ones.

Simulation procedure is the following: firstly the multi-body model has been run as it was the real mechanical system. Basically, it represents the *ground truth*. During such simulation all the data have been recorded in order to compare them with the ones obtained through the non real mechanism. In particular, some of the recorded data have been used to generate virtual sensor measurements, i. e. white noise has been added on them.

After that, a second simulation has been run. This differs from the first for the introduction of some errors in multi-body model representing the non perfect knowledge of actual system. During this second simulation the observer has also be run. The goal is that the observer corrects the multi-body model, which would behave differently with respect to the ground truth, and forces it to follow the recorded performances.

Several errors are introduced into the multi-body model: firstly, there is a difference between ground truth gravity acceleration and multi-body model one, in order to represent a force error due to an unknown model parameter. Ground truth gravity acceleration value is 9.806 $\frac{m}{s^2}$, whereas multi-body model one is chosen 9 $\frac{m}{s^2}$.

Initial condition errors on position and velocity are added too: real mechanism starts with an initial angular position of 60° and a velocity of 1 $\frac{rad}{s}$ clockwise, instead multi-body model initial position and velocity are 90° and 10 $\frac{rad}{s}$ counter-clockwise, respectively. A mass error on the second link – in order to produce an error in the inertia – is introduced: such a link in multi-body model weights 5 kg instead

	Real mechanism	MB model
Applied torque [Nm]	-10 (100)	-10
Initial cond. (q [rad], $\dot{q}$ [$\frac{\mathrm{rad}}{s}$])	1.0472, 1	1.5708, -10
Gravity acceleration [$\frac{m}{s^2}$]	9.806	9
Link 2 mass [kg]	8	5

Table 1: Four-bar linkage simulation parameters.

of 8. Finally, a torque of 110 Nm *acts only on the ground truth* just in the middle of the simulation. This represents an unknown external force which acts suddenly. Simulation time is set to 10 seconds.
Force errors are particularly interesting, since they are the main cause of uncertainty in multi-body simulations. In fact, geometric and mass quantities are usually quite known, on the other hand it is almost impossible to predict all the forces which will act on a mechanism. Regarding to the errors introduced into the multi-body model, apart from initial condition ones, they all produce acceleration errors, leading to position and velocity differences of model variables, with respect to the real system.
After the kinematic initialization, the simulation evolves – at every time-step – as follows:

A. Ground truth dynamics integration;

B. Multi-body model dynamics integration;

C. Observer prediction step;

D. Innovation calculation: comparison of ground truth virtual measurement of crank angle with multi-body model one;

E. Observer updating step;

F. Natural coordinate error calculation through kinematic velocity problem resolution: computation of x_1, y_1, x_2, y_2 errors;

G. Multi-body model correction.

Although several tests were performed, each one with an error, then putting them together, only final results will be shown. Simulation parameters are summed up in Tab. 1. Taking a look to Fig. 14-17, it can be noticed that position estimate, in spite of all the errors introduced into the multi-body model, follows the ground truth, so the observer works correctly. In fact, estimation error of the multi-body model corrected by the observer is negligible, if compared with the one committed by the multi-body model only.
As it was told before, *in this case*, the filter is linear, so the whiteness

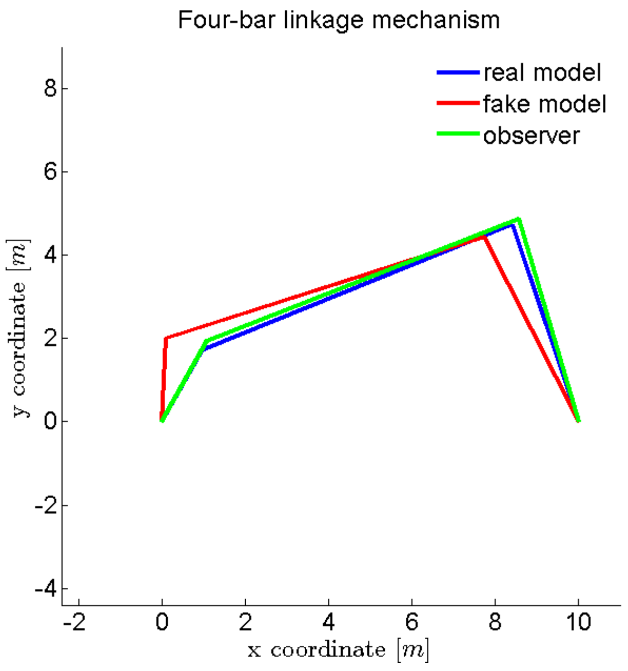

Figure 13: Four-bar linkage mechanism simulation in MATLAB.

test of the estimation error has been used in order to tune up the filter through model noise coefficient modifications. Estimation errors are summed up in Tab. 2: their maximum and mean-squared values, in multi-body model with and without the observer respectively, are listed. Simulation MATLAB code is stated in Appendix A.

3.3.5 *Observer stability, another test*

Another test of observer stability has been performed, in order to confirm it before proceeding to vehicle one design. It is known that an observer has no effect on control system closed-loop pole locations other than to add the poles of the observer itself [19]. A state, which includes the state variables of ground truth $\mathbf{x}$, of multi-body model $\mathbf{x}_{\mathbf{MB}}$, and of observer $\mathbf{e}$, is defined:

$$\mathbf{d} = \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_{\mathbf{MB}} \\ \mathbf{e} \end{bmatrix} \tag{64}$$

The state-transition matrix of the entire system can be determined. Mechanism evolution, like multi-body model one, in state-space form derives from the integration of dynamic equations. Observer evolution is quite different: since the error in prediction step is forced to zero, its state-transition matrix is zero. So, observer evolution is

Coord.	max_{OBS}	max_{MB}	rms_{OBS}	rms_{MB}
x_1 [m]	0.1191	18.915	0.0080	3.6509
y_1 [m]	0.0987	18.151	0.0081	3.4061
x_2 [m]	0.1191	18.915	0.0080	3.6509
y_2 [m]	0.0987	18.151	0.0081	3.4061
$\dot{x}_1$ [$\frac{m}{s}$]	0.8319	9.6530	0.0725	1.6717
$\dot{y}_1$ [$\frac{m}{s}$]	0.8664	10.348	0.0799	1.6616
$\dot{x}_2$ [$\frac{m}{s}$]	0.8319	9.6530	0.0725	1.6717
$\dot{y}_2$ [$\frac{m}{s}$]	0.8664	10.348	0.0799	1.6616

Table 2: Estimation errors in four-bar linkage simulation, respectively of observer and multi-body model.

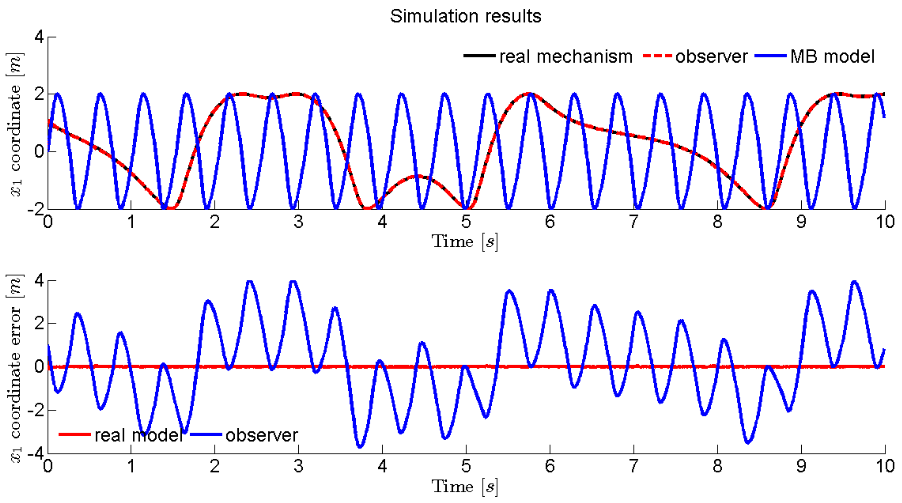

Figure 14: Estimated position of the first joint in x-direction (top). Estimation error (bottom).

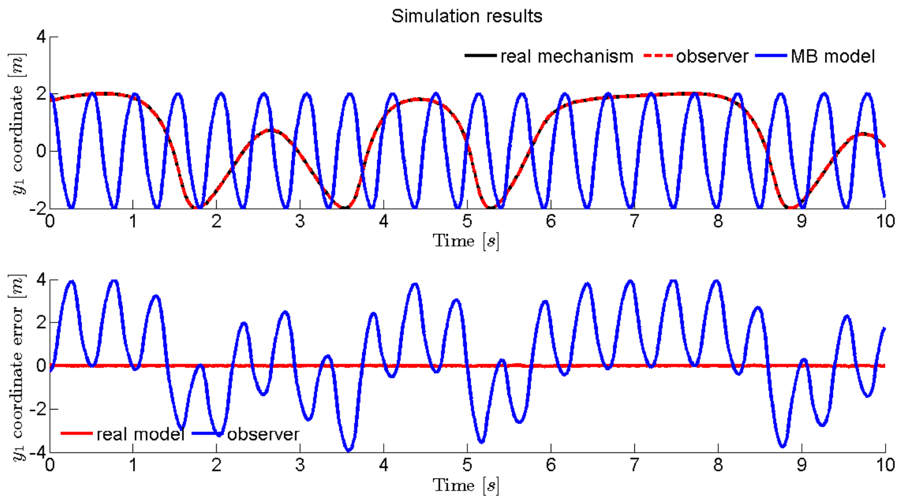

Figure 15: Estimated position of the first joint in y-direction (top). Estimation error (bottom).

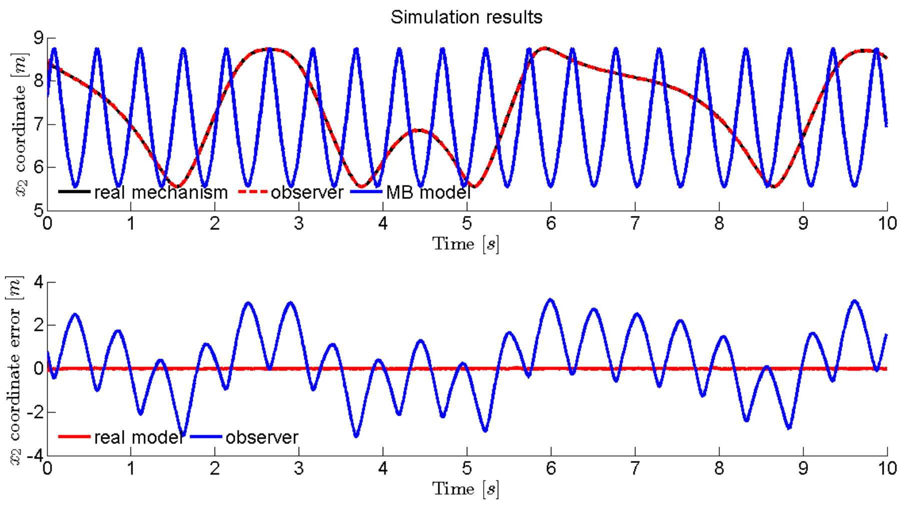

Figure 16: Estimated position of the second joint in x-direction (top). Estimation error (bottom).

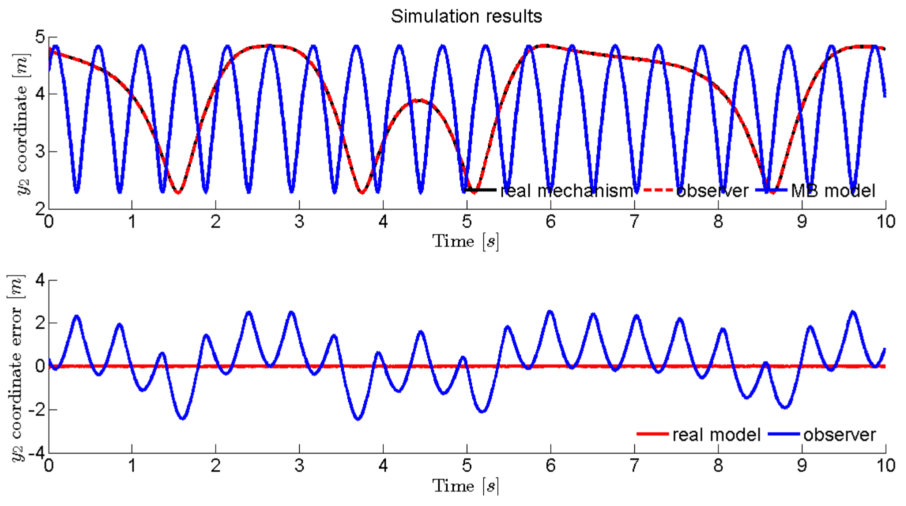

Figure 17: Estimated position of the second joint in y-direction (top). Estimation error (bottom).

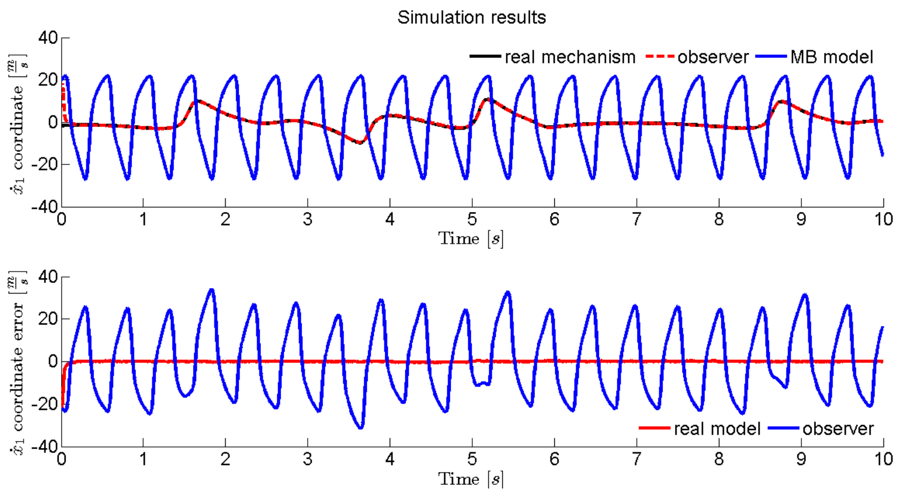

Figure 18: Estimated velocity of the first joint in x-direction (top). Estimation error (bottom).

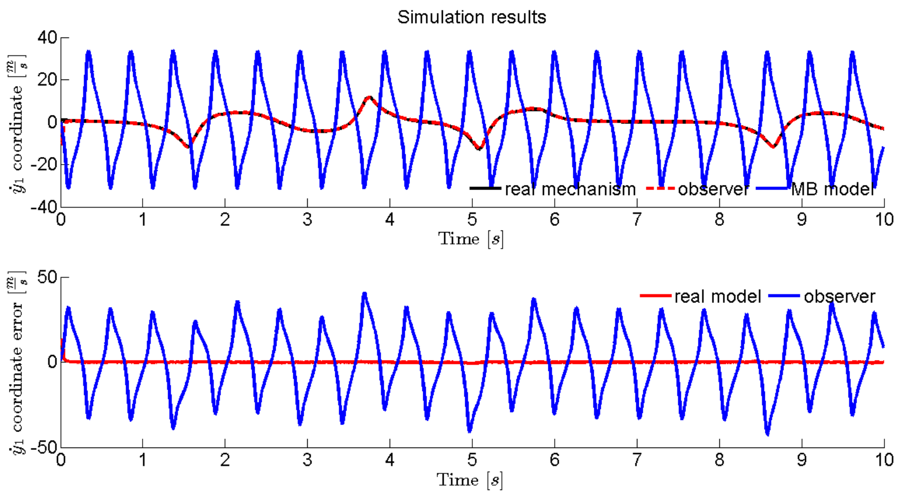

Figure 19: Estimated velocity of the first joint in y-direction (top). Estimation error (bottom).

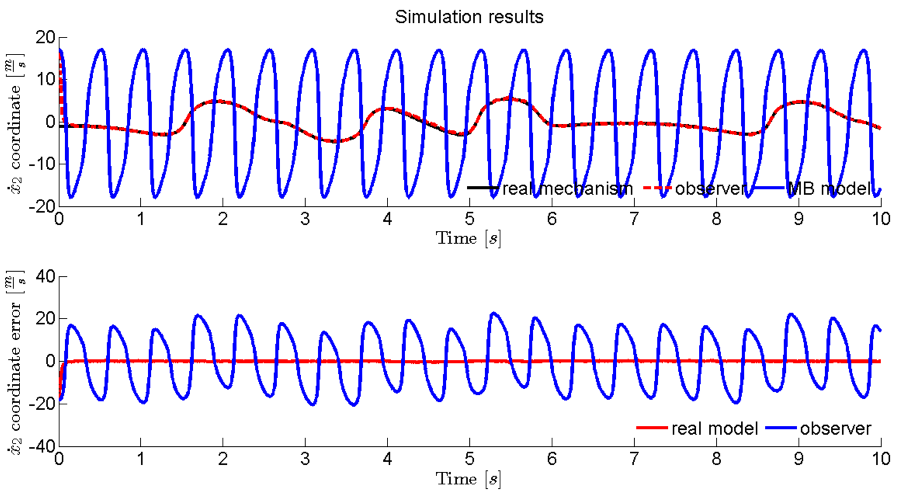

Figure 20: Estimated velocity of the second joint in x-direction (top). Estimation error (bottom).

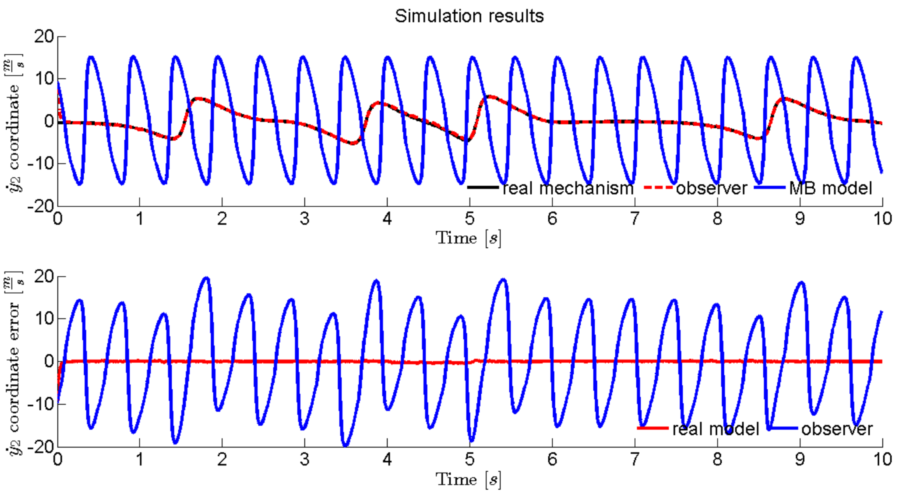

Figure 21: Estimated velocity of the second joint in y-direction (top). Estimation error (bottom).

uniquely determined by mechanism and multi-body model outputs comparison. The result is:

$$\mathbf{d}_{k+1} = \begin{bmatrix} \mathbf{F} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{F_{MB}} & \mathbf{0} \\ \mathbf{FLH} & -\mathbf{F_{MB}LH_{MB}} & \mathbf{0} \end{bmatrix} \mathbf{d}_k \tag{65}$$

In particular, mechanism state-transition matrix can be obtained, in an approximate way, from the linearisation of multi-body model equations [8]:

$$\mathbf{F} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \frac{\partial(\bar{\mathbf{M}}^{-1}\bar{\mathbf{Q}})}{\partial \mathbf{z}} & \frac{\partial(\bar{\mathbf{M}}^{-1}\bar{\mathbf{Q}})}{\partial \dot{\mathbf{z}}} \end{bmatrix} \tag{66}$$

where $\mathbf{z}$, $\dot{\mathbf{z}}$ are independent positions and velocities, whereas accelerations are determined through R-matrix:

$$\ddot{\mathbf{z}} = (\mathbf{R}^T\mathbf{M}\mathbf{R})^{-1}[\mathbf{R}^T(\mathbf{Q} - \mathbf{M}\dot{\mathbf{R}}\dot{\mathbf{z}})] = \bar{\mathbf{M}}^{-1}\bar{\mathbf{Q}} \tag{67}$$

It is well known that, in order to preserve the entire system stability, modules of state-transition matrix eigenvalues must be less than one. So, it has been evaluated during the simulation, verifying that is always less than one. It can be concluded that, under the same hypothesis of traditional Kalman filter stability, the errorEKF maintains itself stable.

4 OBSERVER DESIGN

The four-bar linkage mechanism observer was shown in the previous chapter, now the design of vehicle one will be illustrated. The main changes with respect to the former, apart from the bigger size, are in output matrix calculation. In particular, there is an aspect which must be taken into account, related to the fact that now the degrees of freedom are more than one.

4.1 VEHICLE

Talking about the vehicle, the three attitude angles yaw, pitch and roll will be mentioned. In particular, vehicle observability will be discussed taking into account them, since it results more intuitive. Anyway, it should be reminded that they are not present in the vehicle multi-body model.

4.1.1 *Vehicle simulation*

The vehicle is equipped with a set of sensors which can be divided in two main categories: the ones which are needed in order to make the model run and the ones which measure system outputs. There are, then, other sensors which have an internal function, such as the potentiometer used to measure steering angle in steer-by-wire mechanism. The former ones are two, and measure the two dynamical inputs of the multi-body model: wheel torque and brake pressure.
The torque sensor is mounted on the shaft of rear right wheel, and senses the moment which is applied to the wheels[1]. The brake pressure sensor monitors how the brakes are acting on the vehicle. With these two measurements, the two main internal inputs are known, the others are external and they are approximated or neglected, such as tyre friction and aerodynamic resistance, respectively. The third input of the multi-body model, which is not actually a force, is the *knowledge of the road profile*, from which the force acting on the tyres can be determined. It is worth noticing that, although this assumption is hardly plausible in a commercial available car, it is equivalent to have a load cell on each suspension, in order to sense what in the multi-body model is calculated.

Sensors measuring the system outputs, which will be described more extensively later, are:

1 The equality of left and right wheel torque is assumed.

Measured magnitudes	Sensor	Accuracy	Rate [Hz]
Vehicle accelerations (x,y,z)	Accelerometers	0.5 $[\frac{m}{s^2}]$	500
Vehicle angular rates (x,y,z)	Gyroscopes	0.0002 $[\frac{rad}{s}]$	500
Wheel rotation angles	Hall-effect sensors	0.17 $[rad]$	500
Position (x,y,z)	GPS receiver	0.02 $[m]$	50
Speed	GPS receiver	0.1 $[\frac{m}{s}]$	50
Course over ground	GPS receiver	N.D.	50

Table 3: List of installed sensors.

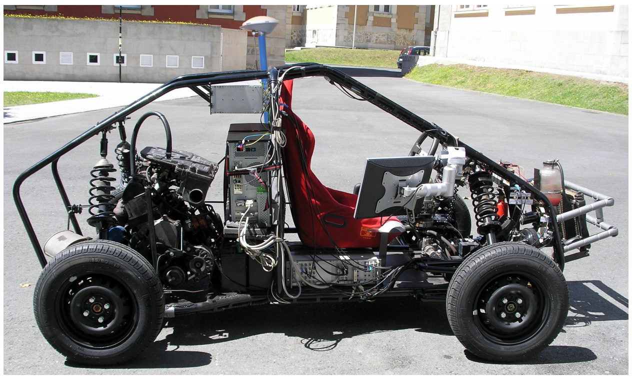

Figure 22: The vehicle.

A. An *Inertial Measurement Unit*[2], composed of a triaxial accelerometer and three gyroscopes;

B. A GPS receiver;

C. Four *Hall effect sensors* mounted on the wheels.

The IMU is behind driver's seat, and provides three angular velocity and acceleration signals, all referred to its axes. GPS antenna is behind driver's seat, too, but it does not have a reference system, so signals are referred to three axes which will be described later. Such a sensor is capable of providing 3D position and 2D velocity of vehicle antenna, so giving great information about chassis movement in the space. The Hall effect sensors behave like an encoder in terms of output signals, so returning wheel angular positions.

The sensors are listed in Tab. 3, where the accuracies are specified. These are the real ones of the sensors mounted on the vehicle: some sets of data were taken and processed, in order to simulate, in the observer code, realistic sensors. A scheme of multi-body simulator working principle is shown in Fig. 23. Of the 168 natural coordinates, fifteen are independent, so they are degrees of freedom of the multi-body model. They are:

2 Usually indicated as IMU.

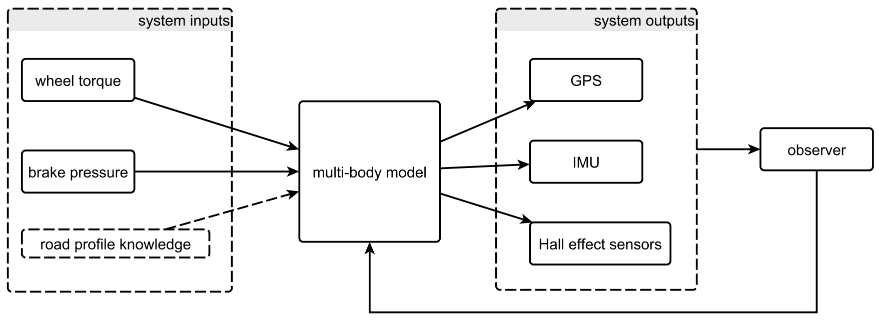

Figure 23: Simulation working principle, showing observer role.

A. The three linear position coordinates x, y, z;

B. The three attitude coordinates $v1_y$, $v1_z$, $v2_z$;

C. The four wheel angles θ_1, θ_2, θ_3, θ_4;

D. The four suspension displacements z_{w_1}, z_{w_2}, z_{w_3}, z_{w_4};

E. The steering angle α.

Since the steering angle is directly controlled by the driver and it is measured for making the steer-by-wire system work, this coordinate is excluded from the observer state vector. All the other fourteen variables are to be observed, so, together with their velocities, they form the observer state vector, whose size is 28.

4.1.2 *Vehicle observability*

It was told in the previous chapter how the substitution of the Jacobian of $f(\mathbf{x}_k)$ function with an integrator, into the filter, leads to a loss of observability. If the classic proof was performed, it would result that the system is fully observable with the sensors which are on board and assuming the knowledge of the road profile.
However, because of the observability loss due to the use of this kind of observer, some modifications have been performed, with respect to what was believed before. In order to perform the observability proof, vehicle model has been divided into three subsystems:

A. Chassis, which can move just rigidly along the three axes x, y, z;

B. Suspensions, characterized by elastic dynamics;

C. Wheels, characterized by angular motions.

The first subsystem leads to the equations:

$$\begin{aligned} \dot{\mathbf{s}}_1 &= \mathbf{F}\mathbf{s}_1 \\ \mathbf{y} &= \mathbf{H}\mathbf{s}_1 \\ \mathbf{s}_1 &:= \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T \end{aligned} \tag{68}$$

where the matrices are:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{69}$$

So, building the observability matrix like in (44), it results:

$$\mathrm{rank}(\mathbf{O}) = \mathrm{rank} \begin{bmatrix} \mathbf{H} \\ \mathbf{HF} \\ \mathbf{HF}^2 \\ \mathbf{HF}^3 \\ \mathbf{HF}^4 \\ \mathbf{HF}^5 \end{bmatrix} = 6 \tag{70}$$

This means that, with position measurements, the rigid dynamics of the vehicle is observable.

Wheel subsystem is very similar: the motion equations are quite the same, wheels have angular displacements instead of linear ones. Moreover, subsystem size is now eight instead of six. However, **F** and **H** matrix structure is the same, this allows skipping the mathematical definition and going directly to the result:

$$\mathrm{rank}(\mathbf{O}) = \mathrm{rank} \begin{bmatrix} \mathbf{H} \\ \mathbf{HF} \\ \mathbf{HF}^2 \\ \mathbf{HF}^3 \\ \mathbf{HF}^4 \\ \mathbf{HF}^5 \\ \mathbf{HF}^6 \\ \mathbf{HF}^7 \end{bmatrix} = 8 \tag{71}$$

This is the confirmation of what was already expected: having a system with the same structure of the one in (68) and measuring all the four angular positions, it was obvious that the subsystem would be result observable.

The last subsystem to analyse is the one related to the elastic dynamic of the vehicle: four vertical suspension displacements and the two pitch and roll angle are taken into account, this leads to:

$$\mathbf{s}_3 := \begin{bmatrix} z_{w_1} & z_{w_2} & z_{w_3} & z_{w_4} & \varphi & \psi & \dot{z}_{w_1} & \dot{z}_{w_2} & \dot{z}_{w_3} & \dot{z}_{w_4} & \dot{\varphi} & \dot{\psi} \end{bmatrix}^T \tag{72}$$

With the available sensors, **F** and **H** matrices result:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{73}$$
$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Thus the outcome of the observability test is the following:

$$\operatorname{rank}(\mathbf{O}) = \operatorname{rank} \begin{bmatrix} \mathbf{H} \\ \mathbf{HF} \\ \mathbf{HF}^2 \\ \mathbf{HF}^3 \\ \mathbf{HF}^4 \\ \mathbf{HF}^5 \\ \mathbf{HF}^6 \\ \mathbf{HF}^7 \\ \mathbf{HF}^8 \\ \mathbf{HF}^9 \\ \mathbf{HF}^{10} \\ \mathbf{HF}^{11} \end{bmatrix} = 2 \tag{74}$$

The subsystem is not observable: remembering what was told about the loss of observability, it is obvious. In fact, since every movement is – from the observer perspective – completely independent from the other ones and suspension displacements are not measured, it is impossible that their dynamics can be observed.
Fr such a reason suspension dynamics have been eliminated from observer state vector. This means that their behaviour will follow multibody model one, and it will not be corrected. This could not be a big problem: usually suspension model – for automotive manufacturers – is quite known, so it does not represent one of the main uncertainty factors.
Only pitch and roll angles have been left into observer state vector. Since the IMU contains a triaxial accelerometer which has not been used yet, a method for extracting roll and pitch information from it was thought up, and it will be discussed later.
Of all the vehicle degrees of freedom, only the yaw angle has not been considered yet. Anyway, a trick for making it observable was found out, and it will be explained in the following paragraph. Basically, a yaw measurement[3] is provided to the filter, taken from GPS signal.
Therefore, subsystem state vector becomes:

$$\mathbf{s}_3 := \begin{bmatrix} \phi & \psi & \lambda & \dot{\phi} & \dot{\psi} & \dot{\lambda} \end{bmatrix}^T \tag{75}$$

3 Actually, it will be seen that not the yaw angle measurement is provided to the filter but something similar.

As a consequence, all the three attitude angle measurements are available, and since the three angular velocities[4] are measured, **F** and **H** matrices can be rewritten as:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{78}$$

No calculation is needed for finding out that the subsystem is fully observable, since every state component is measured. So, a stable covariance matrix is expected.

The full state vector, which contains the variables whose estimation error will be determined, is:

$$\mathbf{s} := \begin{bmatrix} x & y & z & \phi & \psi & \lambda & \theta_1 & \theta_2 & \theta_3 & \theta_4 & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\psi} & \dot{\lambda} & \dot{\theta}_1 & \dot{\theta}_2 & \dot{\theta}_3 & \dot{\theta}_4 \end{bmatrix}^T \tag{79}$$

A particular aspect should be underlined: actually, since not all the measurements required are available at every time step – remember that GPS works at 50 Hz – the filter *is not* always observable: observability is lost when not all the sensors are working. So, a covariance growth can be expected in those instants, then its decrease when all the measurements are available. In this case, GPS provides information

4 Actually, the three sensed angular velocities *are not always equal* to roll, pitch and yaw velocities. Generally speaking, in the 3D case, their relationship with them is:

$$\begin{aligned} \dot{\psi} &= [\omega_y \sin(\psi) + \omega_z \cos(\psi)] \tan(\lambda) + \omega_x \\ \dot{\lambda} &= \omega_y \cos(\psi) - \omega_z \sin(\psi) \\ \dot{\phi} &= \frac{\omega_y \sin(\psi) + \omega_z \cos(\psi)}{\cos(\lambda)} \end{aligned} \tag{76}$$

For small attitude angles:

$$\begin{aligned} \dot{\psi} &\approx \omega_x \\ \dot{\lambda} &\approx \omega_y \\ \dot{\phi} &\approx \omega_z \end{aligned} \tag{77}$$

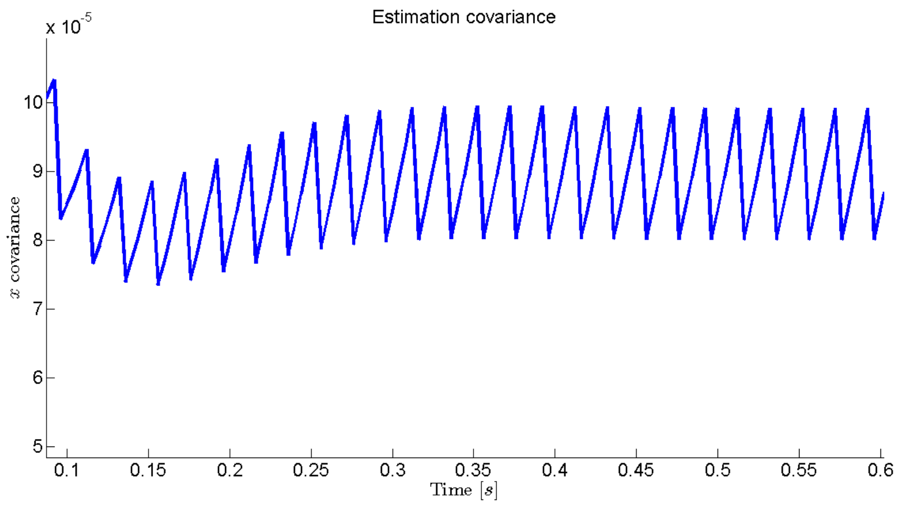

Figure 24: Estimation covariance of x position, zoomed in order to see the sawtooth behaviour.

every five time-steps, whereas all other sensors work at a frequency which is double with respect to multi-body model one. So, a sawtooth behaviour of GPS-related covariances is expected, as it can be seen in Fig. 24.

The last thing which needs to be underlined is that, although a *one* was put in the output matrices, the exact measurements will not be provided. In fact, since some tricks will be performed, the situation will be slightly different. Anyway, it will be seen that, if the sensors are trusted enough, no changes in observability are faced.

4.2 GPS

4.2.1 *GPS considerations*

A lot of people in everyday life use GPS technology, in smart-phones just for localization or in car navigation devices to have the route indicated. A GPS receiver calculates its antenna position measuring the distance between it and a constellation of satellites. This distance is given by determination of the signal *time of flight* , from satellite emission instant until antenna reception one.
Normally, positioning errors can be obtained because of, for example, clock drifts, satellite position errors, atmospheric effects. If two receivers are available, and one of them is at a *known location*, it can be used in order to improve the accuracy, through the measurement of positioning errors. This technique, called *differential* GPS, is used in buggy device, and it allows reaching 2 centimetres accuracy.

Moreover, a GPS receiver can work at a frequency up to 50 Hz and can provide velocity measurements, based on Doppler effect[5].

Satellites are located in the WGS84 reference system, which provides a mathematical Earth model from a geodetic, geometric and gravitational point of view. GPS positioning output consist of three variables, all expressed in this reference system: longitude, latitude, and ellipsoidal heigh – see Fig. 25 – respectively expressed as λ, ϕ, h. Actually, in this work the corrections suggested by the IGN[6] were used, so moving the reference system from WGS84 to ETRS89 one, which moves with the Eurasian plate. In this, the geodetic parameters of latitude and heigh are defined in terms of the ellipsoid normal at the user's position.
In order to be used along with the multi-body model, a set of coordinates which place GPS position with respect to the same axes of the multi-body simulation is preferred. So, a transformation to ENU[7] reference system is carried out in two steps: firstly a transformation from geodetic to ECEF[8] reference system is done:

$$\mathbf{r}^{\mathbf{ECEF}} = \begin{bmatrix} \dfrac{a\cos(\lambda)}{\sqrt{1+(1-e^2)\tan^2(\phi)}} + h\cos(\lambda)\cos(\phi) \\ \dfrac{a\sin(\lambda)}{\sqrt{1+(1-e^2)\tan^2(\phi)}} + h\sin(\lambda)\cos(\phi) \\ \dfrac{a(1-e^2)\sin\phi}{\sqrt{1-e^2\sin^2(\phi)}} + h\sin(\phi) \end{bmatrix} \tag{80}$$

being a the semi-major axis of the ETRS89 ellipsoid, b its semi-minor axis, and e the ellipsoid eccentricity:

$$\begin{aligned} a &= 6378137.000 \text{ m} \\ b &= 6356752.314140 \text{ m} \\ e &= \sqrt{1-\frac{b^2}{a^2}} \end{aligned} \tag{81}$$

After that, the transformation from ECEF to ENU in performed, according to the following:

$$\begin{aligned} \mathbf{r}^{\mathbf{ENU}} &= \mathbf{W}(\mathbf{r}_{\mathbf{P}}^{\mathbf{ECEF}} - \mathbf{r}_{\mathbf{0}}^{\mathbf{ECEF}}) \\ \mathbf{W} &= \begin{bmatrix} -\sin(\lambda) & \cos(\lambda) & 0 \\ \cos(\lambda)\sin(-\phi) & \sin(\lambda)\sin(-\phi) & \cos(-\phi) \\ \cos(\lambda)\cos(-\phi) & \sin(-\phi)\cos(-\phi) & -\sin(-\phi) \end{bmatrix} \end{aligned} \tag{82}$$

5 Modern GPS devices, such as GT-11, implement PLL receivers which continuously track satellite signal carriers. The difference between the known satellite carrier frequency and the one determined at the receiver is known as a *Doppler shift*. This is directly proportional to receiver velocity along satellite direction, regardless of its distance.

6 Instituto Geográfico Nacional.

7 East, North, Up.

8 Earth-Centered, Earth-Fixed.

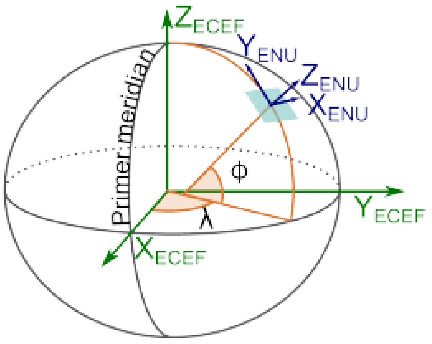

Figure 25: ECEF and ENU reference systems.

where $\mathbf{r}_\mathbf{P}^{\mathbf{ECEF}}$ is the position of the GPS receiver and $\mathbf{r}_\mathbf{0}^{\mathbf{ECEF}}$ is the position of the origin of the ENU system.

4.2.2 *GPS position model*

As it was told, the GPS receiver provides antenna position and velocity measurements. So, these are the measurement functions which are to be modelled, and from which the output function Jacobian is to be calculated.
The antenna is located behind driver's head, and its coordinates expressed with respect to chassis point 1 are called x_{GPS}, y_{GPS}, z_{GPS}. So, regarding to position measurements, the model can be formulated as follows:

$$\mathbf{r_P} = \mathbf{r_{CH}} + x_{GPS}\mathbf{v_1} + y_{GPS}\mathbf{v_2} + z_{GPS}\mathbf{v_3} \tag{83}$$

where $\mathbf{r_P}$ is GPS antenna position expressed in the global reference frame, whereas $\mathbf{r_{CH}}$ is chassis point 1 position, in the same reference frame. $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$ are the three unit vectors used to model vehicle chassis. This equation has to be differentiated with respect to the independent variables of the system. It can be re-written as:

$$\begin{bmatrix} x_P \\ y_P \\ z_P \end{bmatrix} = \begin{bmatrix} x_{CH} \\ y_{CH} \\ z_{CH} \end{bmatrix} + x_{GPS} \begin{bmatrix} v1_x \\ v1_y \\ v1_z \end{bmatrix} + y_{GPS} \begin{bmatrix} v2_x \\ v2_y \\ v2_z \end{bmatrix} + z_{GPS} \begin{bmatrix} v3_x \\ v3_y \\ v3_z \end{bmatrix} \tag{84}$$

It can be noticed that:

$$\frac{\partial \mathbf{r_P}}{\partial \mathbf{z}} = \frac{\partial \mathbf{r_P}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{z}} = \frac{\partial \mathbf{r_P}}{\partial \mathbf{q}} \mathbf{R} \tag{85}$$

where R-matrix, already mentioned in the previous paragraph, contains the relationships between the infinitesimal displacement of each dependent coordinate and the one of each degree of freedom. Since these equations only depend on chassis point 1 position and on the three unit vectors, the partial derivatives with respect to the other natural coordinates is zero. The differentiation can be limited to:

$$\frac{\partial \mathbf{r_P}}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial x_P}{\partial x} & \frac{\partial x_P}{\partial y} & \frac{\partial x_P}{\partial z} & \cdots & \frac{\partial x_P}{\partial \theta_4} \\ \frac{\partial y_P}{\partial x} & \frac{\partial y_P}{\partial y} & \frac{\partial y_P}{\partial z} & \cdots & \frac{\partial y_P}{\partial \theta_4} \\ \frac{\partial z_P}{\partial x} & \frac{\partial z_P}{\partial y} & \frac{\partial z_P}{\partial z} & \cdots & \frac{\partial z_P}{\partial \theta_4} \end{bmatrix} \tag{86}$$

Taking a look to the model in (84), it can be seen that GPS receiver antenna position is a linear function of the one of chassis point 1 and of the three unit vectors. So, apart from these, every other term in (85) will be zero. Considering the partial derivatives with respect to the three unit vectors, it results:

$$\begin{gathered} \frac{\partial x_P}{\partial v_{1_x}} = x_{GPS} \quad \frac{\partial x_P}{\partial v_{2_x}} = y_{GPS} \quad \frac{\partial x_P}{\partial v_{3_x}} = z_{GPS} \\ \frac{\partial y_P}{\partial v_{1_y}} = x_{GPS} \quad \frac{\partial y_P}{\partial v_{2_y}} = y_{GPS} \quad \frac{\partial y_P}{\partial v_{3_y}} = z_{GPS} \\ \frac{\partial z_P}{\partial v_{1_z}} = x_{GPS} \quad \frac{\partial z_P}{\partial v_{2_z}} = y_{GPS} \quad \frac{\partial z_P}{\partial v_{3_z}} = z_{GPS} \\ \frac{\partial x_P}{\partial v_{1_y}} = \frac{\partial x_P}{\partial v_{1_z}} = \cdots = \frac{\partial z_P}{\partial v_{3_x}} = \frac{\partial z_P}{\partial v_{3_y}} = 0 \end{gathered} \tag{87}$$

It is worth noticing, anyway, that the aim of these information is that the filter can correct vehicle position. So, putting these partial differences into the output matrix, the observer will know that it has to collocate the antenna in the right place. However, it would be desirable to have *the vehicle chassis* well-positioned, instead of the antenna. Correcting antenna position the observer could make the chassis rotate about its three axes.

Metaphorically speaking, the situation is equal to one in which a big hand could bring the vehicle for displacing it – as it can be seen in Fig. 26 – in order to correct its position. Would it be preferable to have it grabbed by the chassis or by the antenna? The second solution was thought to be the most correct. So the partial derivatives of position model with respect to the three unit vectors were neglected, even though, mathematically, to consider them would have been more rigorous.

Obviously, it is:

$$\begin{gathered} \frac{\partial x_P}{\partial x} = 1 \\ \frac{\partial y_P}{\partial y} = 1 \\ \frac{\partial z_P}{\partial z} = 1 \\ \frac{\partial x_P}{\partial y} = \frac{\partial x_P}{\partial z} = \frac{\partial y_P}{\partial x} = \frac{\partial y_P}{\partial z} = \frac{\partial z_P}{\partial x} = \frac{\partial z_P}{\partial y} = 0 \end{gathered} \tag{88}$$

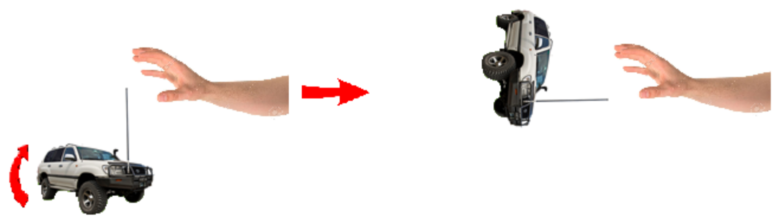

Figure 26: Effect of antenna position correction, instead of chassis one.

This leads to:

$$\frac{\partial \mathbf{r_P}}{\partial \mathbf{q}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{89}$$

which is precisely what it was needed.
Since that this matrix is linear with respect to the state, a behaviour close to the optimum should be expected here. Anyway, if an unknown external constant force were applied, innovation spectrum could not be white unless that external force was included into the model. If a front wind is imagined, a constant force opposite to the vehicle motion is acting on the system. So, the error, for example, in x coordinate would always have the same sign, this does not allows obtaining white innovation.

4.2.3 *GPS velocity model*

Regarding to velocity, the model can be obtained by differentiating equation (84) with respect to the time. Yet the GPS does not provide a velocity signal along z axis, so leading to:

$$\begin{bmatrix} \dot{x}_P \\ \dot{y}_P \end{bmatrix} = \begin{bmatrix} \dot{x}_{CH} \\ \dot{y}_{CH} \end{bmatrix} + x_{GPS} \begin{bmatrix} \dot{v}_{1_x} \\ \dot{v}_{1_y} \end{bmatrix} + y_{GPS} \begin{bmatrix} \dot{v}_{2_x} \\ \dot{v}_{2_y} \end{bmatrix} + z_{GPS} \begin{bmatrix} \dot{v}_{3_x} \\ \dot{v}_{3_y} \end{bmatrix} \tag{90}$$

It is worth noticing that the velocity information are expressed in polar coordinates, that is, the module of velocity vector and the course over ground angle with respect to the local ENU north. However, with a simple transformation to Cartesian coordinates, equation (90) can be obtained. In this work, it was done off-line using MATLAB, yet the introduction into the observer code of this calculation would not be a burden for the programmer nor a big effort for the processor.
The calculation is very similar to position one:

$$\frac{\partial \dot{\mathbf{r}}_\mathbf{P}}{\partial \dot{\mathbf{z}}} = \frac{\partial \dot{\mathbf{r}}_\mathbf{P}}{\partial \dot{\mathbf{q}}} \frac{\partial \dot{\mathbf{q}}}{\partial \dot{\mathbf{z}}} \tag{91}$$

where, again, the second term in equation right side is R-matrix, which relates dependent coordinates infinitesimal velocities to independent coordinates ones. The partial derivatives with respect to chassis point 1 velocity are quite the same of position ones:

$$\begin{aligned}
&\frac{\partial \dot{x}_P}{\partial \dot{x}} = 1 \\
&\frac{\partial \dot{y}_P}{\partial \dot{y}} = 1 \\
&\frac{\partial \dot{x}_P}{\partial \dot{y}} = \frac{\partial \dot{x}_P}{\partial \dot{z}} = \frac{\partial \dot{y}_P}{\partial \dot{x}} = \frac{\partial \dot{y}_P}{\partial \dot{z}} = 0
\end{aligned} \tag{92}$$

The partial derivatives with respect to the three unit vectors velocity are analogous too, taking into account that some of them are not present, since $\dot{z}_P$ measurement is not provided by the GPS.

$$\begin{aligned}
&\frac{\partial \dot{x}_P}{\partial \dot{v}_{1_x}} = x_{GPS} \quad \frac{\partial \dot{x}_P}{\partial \dot{v}_{2_x}} = y_{GPS} \quad \frac{\partial \dot{x}_P}{\partial \dot{v}_{3_x}} = z_{GPS} \\
&\frac{\partial \dot{y}_P}{\partial \dot{v}_{1_y}} = x_{GPS} \quad \frac{\partial \dot{y}_P}{\partial \dot{v}_{2_y}} = y_{GPS} \quad \frac{\partial \dot{y}_P}{\partial \dot{v}_{3_y}} = z_{GPS} \\
&\frac{\partial \dot{x}_P}{\partial \dot{v}_{1_y}} = \frac{\partial \dot{x}_P}{\partial \dot{v}_{2_y}} = \cdots = \frac{\partial \dot{y}_P}{\partial \dot{v}_{2_x}} = \frac{\partial \dot{y}_P}{\partial \dot{v}_{3_x}} = 0
\end{aligned} \tag{93}$$

In this case the reason for neglecting these terms would be the same one used for position problem. Anyway, it would be more correct to take into account the difference in velocities between chassis and antenna, so considering them. This because, for example, if the car passed over a speed bump, antenna tangential velocity would suddenly change, although chassis linear one would be the same. Neglecting these terms would mean, in a case like this, correct chassis linear velocity even though there is nothing to be corrected. So, the choice was to introduce these terms into the matrix, so leading to:

$$\frac{\partial \dot{\mathbf{r}}_{\mathbf{P}}}{\partial \dot{\mathbf{q}}} = \begin{bmatrix} 1 & 0 & \dots & x_{GPS} & 0 & 0 & y_{GPS} & 0 & 0 & z_{GPS} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & x_{GPS} & 0 & 0 & y_{GPS} & 0 & 0 & z_{GPS} & \dots & 0 \end{bmatrix} \tag{94}$$

After the introduction of these terms, this part of the observer is no more linear with respect to the state, so a behaviour which is farer from the optimum, with respect to position part, is expected.

4.2.4 *Yaw angle*

It was told the three Tait-Bryan angles yaw, pitch, roll are not present in multi-body models in natural coordinates, since this kind of notation does not use angles. How the rigid body rotations are expressed was explained, and it was showed in which way this was applied to vehicle model.
During observer design, however, a problem was faced: imagining the vehicle running in the direction of global reference frame x axis,

a module variation of $\mathbf{v_1}$ unit vector y component makes the vehicle rotate about its z axis. Yet if the vehicle turns, aligning itself to global reference frame y axis, that variation does not make the vehicle rotate, since it is parallel to its running direction.
This is the reason why the *yaw* angular variable ϕ was introduced: with this modification the just explained problem is no more present. Remembering natural coordinate notation, a *restriction* has to be introduced in order to add the variable. It was modelled as follows:

$$\phi = atan2(v_{1_y}, v_{1_x}) \tag{95}$$

In order to correct vehicle yaw angle, a measurement of ϕ must be provided to the observer. Actually, no sensor which is capable of measuring it is present on board, so the *course over ground* GPS information was used as it was the yaw angle. It is worth pointing out that the two angles can be very different: the more the vehicle is turning, the more the COG is different from the yaw. Anyway, it is the best thing it was possible to do.
Since COG measurement is provided to the filter *as it was yaw measurement*, the output matrix will simply have a *one* at the correct position. This leads to:

$$\frac{\partial \phi}{\partial \mathbf{z}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix} \tag{96}$$

It is worth noticing, moreover, that although this output matrix part is linear, actually the observer is *cheated* accepting information believing it was something else. So, it was already known an optimum behaviour of the filter was not to be expected, yet this part behaviour will be farer from the optimum than position part was.
Another thing must be taken into account: the COG signal depends on velocity, so it is not reliable when the vehicle is not moving. So, a velocity check was introduced into the code, in order to discern if this measurement was to be trusted or not. In pseudo-code:

```
% Check if the vehicle is moving or not
if atan2(y velocity,x velocity) > minimum velocity

        % If the vehicle is moving, so COG measurement is
            reliable
        COG is available
else

        COG is NOT available
end
```

Thank to the *scalar gain calculation*, which will be shown later, it is sufficient to change the COG *availability* flag in order to indicate if it is reliable or not, so if the filter will receive this measurement or not.

4.3 INERTIAL MEASUREMENT UNIT

An IMU[9] is mounted on board, like in every commercial available vehicle. Three MEMS gyroscopes and a triaxial MEMS accelerometer are inside, from which information about chassis dynamics have been taken out.

4.3.1 *Gyroscopes model*

A *gyroscope* is a rotating device which, according to angular momentum conservation law, attempts to maintain its axis aligned to a constant direction. Due to this, it is suitable for maintaining or measuring a rotation. A *vibrating gyroscope*[10] is a sensor which measures an angular rate through its rotating structure. As the name suggests, its working principle is based on Coriolis effect.
Considering two proof masses m vibrating in a plane at a frequency ω_r, Coriolis acceleration induced on the two masses is:

$$\mathbf{a_C} = 2(\mathbf{v} \times \mathbf{\Omega}) \tag{97}$$

where $\mathbf{v}$ is a linear velocity and ˙ is the angular rate. The in-plane vibrating displacement is:

$$\mathbf{x_{ip}} = X_{ip} \sin(\omega_r t) \tag{98}$$

So the in-plane velocity is:

$$\dot{\mathbf{x}}_{\mathbf{ip}} = X_{ip} \omega_r \cos(\omega_r t) \tag{99}$$

So, named y_{op} the out-of-plane displacement induced by rotations and F_C the Coriolis-effect force, it follows:

$$y_{op} = \frac{F_C}{k_{op}} = \frac{2m\Omega X_{ip} \omega_r \cos(\omega_r t)}{k_{op}} \tag{100}$$

where k_{op} is a spring constant in the out-of-plane direction and Ω is the magnitude of a rotation vector in the plane of and perpendicular to the driven proof mass motion. So, by measuring the displacement y_{op}, the angular rate Ω can be determined.

A gyroscope can be modelled, given a local orthonormal triad of axes $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$, as a function returning an angular velocity:

$$\boldsymbol{\omega}_{\mathbf{loc}} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \mathbf{k}^T \dot{\mathbf{j}} \\ \mathbf{i}^T \dot{\mathbf{z}} \\ \mathbf{j}^T \dot{\mathbf{i}} \end{bmatrix} \tag{101}$$

9 Inertial Measurement Unit
10 The name was standardized by IEEE as *Coriolis Vibratory Giroscope.*

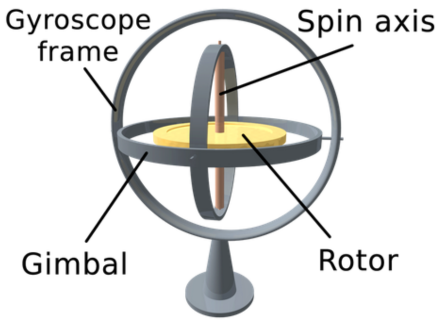

Figure 27: Gyroscope.

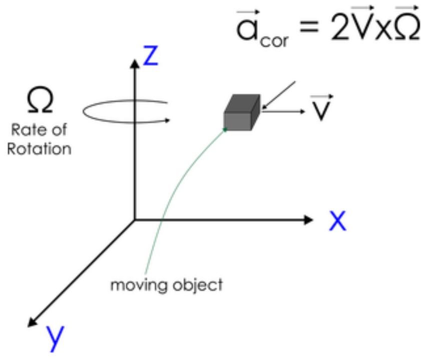

Figure 28: Coriolis force acting on a mass moving in a rotating reference frame.

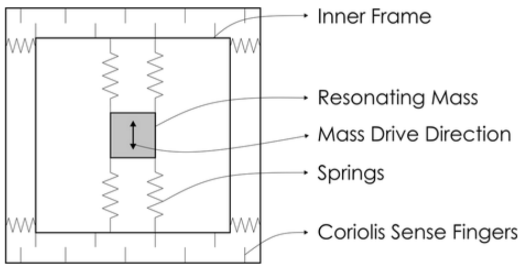

Figure 29: MEMS vibrating gyroscope structure scheme.

Regarding to this application, the model can be applied substituting:

$$\begin{aligned} \mathbf{i} &= \mathbf{v_1} \\ \mathbf{j} &= \mathbf{v_2} \\ \mathbf{k} &= \mathbf{v_3} \end{aligned} \tag{102}$$

And then taking care, in order to obtain the angular velocities expressed with respect to IMU reference frame, of multiplying by the rotation **T** matrix transposed:

$$\boldsymbol{\Omega} = \mathbf{T}^T \boldsymbol{\omega_{loc}} \tag{103}$$

In fact, **T** matrix expresses IMU axes with respect to chassis ones. So, in order to perform the inverse transformation, this matrix can be simply transposed[11]. Everything can be re-written as:

$$\boldsymbol{\Omega} = \mathbf{T}^T \boldsymbol{\omega_{loc}} = \mathbf{T}^T \begin{bmatrix} \mathbf{v_3}^T \dot{\mathbf{v}}_2 \\ \mathbf{v_1}^T \dot{\mathbf{v}}_3 \\ \mathbf{v_2}^T \dot{\mathbf{v}}_1 \end{bmatrix} = \mathbf{T}^T \begin{bmatrix} \begin{bmatrix} v3_x & v3_y & v3_z \end{bmatrix} \begin{bmatrix} \dot{v2}_x \\ \dot{v2}_y \\ \dot{v2}_z \end{bmatrix} \\ \begin{bmatrix} v1_x & v1_y & v1_z \end{bmatrix} \begin{bmatrix} \dot{v3}_x \\ \dot{v3}_y \\ \dot{v3}_z \end{bmatrix} \\ \begin{bmatrix} v2_x & v2_y & v2_z \end{bmatrix} \begin{bmatrix} \dot{v1}_x \\ \dot{v1}_y \\ \dot{v1}_z \end{bmatrix} \end{bmatrix} \tag{104}$$

When computing the partial derivatives, it is worth noticing that the rotation matrix **T** is constant, so it can be excluded from the calculation. It results:

$$\begin{aligned} &\frac{\partial \omega_x}{\partial v_{3_x}} = \dot{v}_{2_x} \quad \frac{\partial \omega_x}{\partial v_{3_y}} = \dot{v}_{2_y} \quad \frac{\partial \omega_x}{\partial v_{3_z}} = \dot{v}_{2_z} \\ &\frac{\partial \omega_x}{\partial \dot{v}_{2_x}} = v_{3_x} \quad \frac{\partial \omega_x}{\partial \dot{v}_{2_y}} = v_{3_y} \quad \frac{\partial \omega_x}{\partial \dot{v}_{2_z}} = v_{3_z} \\ &\frac{\partial \omega_y}{\partial v_{1_x}} = \dot{v}_{3_x} \quad \frac{\partial \omega_y}{\partial v_{1_y}} = \dot{v}_{3_y} \quad \frac{\partial \omega_y}{\partial v_{1_z}} = \dot{v}_{3_z} \\ &\frac{\partial \omega_y}{\partial \dot{v}_{3_x}} = v_{1_x} \quad \frac{\partial \omega_y}{\partial \dot{v}_{3_y}} = v_{1_y} \quad \frac{\partial \omega_y}{\partial \dot{v}_{3_z}} = v_{1_z} \\ &\frac{\partial \omega_z}{\partial v_{2_x}} = \dot{v}_{1_x} \quad \frac{\partial \omega_z}{\partial v_{2_y}} = \dot{v}_{1_y} \quad \frac{\partial \omega_z}{\partial v_{2_z}} = \dot{v}_{1_z} \\ &\frac{\partial \omega_z}{\partial \dot{v}_{1_x}} = v_{2_x} \quad \frac{\partial \omega_z}{\partial \dot{v}_{1_y}} = v_{2_y} \quad \frac{\partial \omega_z}{\partial \dot{v}_{1_z}} = v_{2_z} \end{aligned} \tag{105}$$

This leads to output matrix determination:

$$\mathbf{H_{gyro}} = \begin{bmatrix} \frac{\partial \boldsymbol{\Omega}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{z}} + \frac{\partial \boldsymbol{\Omega}}{\partial \dot{\mathbf{q}}} \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{z}} & \frac{\partial \boldsymbol{\Omega}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \dot{\mathbf{z}}} + \frac{\partial \boldsymbol{\Omega}}{\partial \dot{\mathbf{q}}} \frac{\partial \dot{\mathbf{q}}}{\partial \dot{\mathbf{z}}} \end{bmatrix} \tag{106}$$

11 Since the axes triad is orthonormal, the inverted matrix is equal to the transposed one.

where $\frac{\partial \mathbf{q}}{\partial \dot{\mathbf{z}}}$ is zero. Since it is:

$$\frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{z}} = \mathbf{R_q} \mathbf{R} \dot{\mathbf{z}} = \mathbf{R_q} \dot{\mathbf{q}} = \dot{\mathbf{R}} \tag{107}$$

where $\mathbf{R_q}$ is R-matrix Jacobian and $\dot{\mathbf{R}}$ is its time derivative, this leads to:

$$\mathbf{H_{gyro}} = \begin{bmatrix} \frac{\partial \boldsymbol{\Omega}}{\partial \mathbf{q}} \mathbf{R} + \frac{\partial \boldsymbol{\Omega}}{\partial \dot{\mathbf{q}}} \dot{\mathbf{R}} & \frac{\partial \boldsymbol{\Omega}}{\partial \dot{\mathbf{q}}} \mathbf{R} \end{bmatrix} \tag{108}$$

The matrices in (108) can be calculated as:

$$\begin{aligned} \frac{\partial \boldsymbol{\Omega}}{\partial \mathbf{q}} &= \mathbf{T}^T \begin{bmatrix} 0 & \dots & 0 & 0 & 0 & 0 & \dot{v}_{2_x} & \dot{v}_{2_y} & \dot{v}_{2_z} & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dot{v}_{3_x} & \dot{v}_{3_y} & \dot{v}_{3_z} & 0 & \dots & 0 \\ 0 & \dots & 0 & \dot{v}_{1_x} & \dot{v}_{1_y} & \dot{v}_{1_z} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix} \\ \frac{\partial \boldsymbol{\Omega}}{\partial \dot{\mathbf{q}}} &= \mathbf{T}^T \begin{bmatrix} 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v_{3_x} & v_{3_y} & v_{3_z} & 0 & \dots & 0 \\ 0 & \dots & 0 & v_{1_x} & v_{1_y} & v_{1_z} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & v_{2_x} & v_{2_y} & v_{2_z} & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix} \end{aligned} \tag{109}$$

Here, a highly sub-optimal behaviour of the observer is expected, since output matrix is non-linear with respect to the state.

4.3.2 *Accelerometer model*

An *accelerometer* is a sensor which is capable of providing information about the acceleration of an object it is installed on. Several kinds of accelerometers are commercially available, yet they all share the same operating principle: a proof mass lying on an elastic element is moving because of acceleration and a position transducer measures its displacement. So, knowing mass inertia, it is possible to determine acceleration.
Since gravity is always acting on the mass, even though it is not moving, device response is an acceleration equal to g upwards in quiet conditions. So, in order to obtain the correct acceleration of the object, that quantity must be subtracted. Moreover, gravity influence can be useful to find out object orientation: when it is inclined of an angle β, gravity contribution will be a product of $\mathbf{g}$ vector multiplied by some angle trigonometric function. This is the reason why the accelerometers are used as orientation sensors [21]. In particular, in this case roll and pitch angle must be measured in order to maintain the filter stable, even if the measurements were bad. So, extracting this orientation information from accelerometers, the aim was reached.
Accelerometer model can be formulated as follows:

$$\mathbf{a_{glob}} = \mathbf{a_{CH}} + x_{IMU} \ddot{\mathbf{v}}_1 + y_{IMU} \ddot{\mathbf{v}}_2 + z_{IMU} \ddot{\mathbf{v}}_3 - \mathbf{g} \tag{110}$$

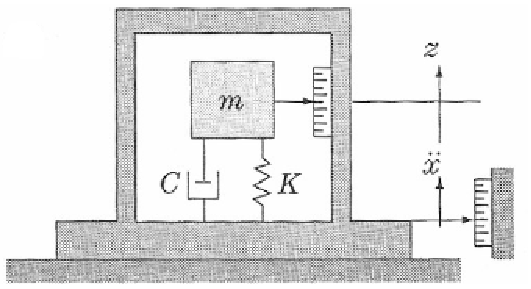

Figure 30: Seismic mass accelerometer.

where $\mathbf{a_{CH}}$ is chassis point 1 acceleration and x_{IMU}, y_{IMU}, z_{IMU} are IMU coordinates with respect to that point. In order to obtain acceleration value expressed in the local reference frame, $\mathbf{a_{glob}}$ vector must be multiplied by chassis rotation matrix $\mathbf{C}$ transposed. In fact, this is the rotation matrix which expresses chassis orientation in the global reference frame:

$$\mathbf{C} = \begin{bmatrix} v1_x & v2_x & v3_x \\ v1_y & v2_y & v3_y \\ v1_z & v2_z & v3_z \end{bmatrix} \tag{111}$$

In order to model accelerometer response the transformation has to be inverted. Moreover, IMU axes have to be aligned to that point ones, through the multiplication by the rotation matrix $\mathbf{T}$ transposed, already mentioned in the previous paragraph. In fact, these two rotations must be composed in order to obtain accelerometer model, so that composition must be inverted:

$$\mathbf{a_{IMU}} = (\mathbf{CT})^T \mathbf{a_{glob}} = \mathbf{T}^T \mathbf{C}^T \mathbf{a_{glob}} \tag{112}$$

As told in the previous paragraph, since the axis triads are orthonormal, it is sufficient to transpose the matrices in order to invert the transformation. This leads to:

$$\mathbf{a_{glob}} = \mathbf{T}^T \begin{bmatrix} v1_x & v1_y & v1_z \\ v2_x & v2_y & v2_z \\ v3_x & v3_y & v3_z \end{bmatrix} \mathbf{a_{loc}} \tag{113}$$

Once accelerometer response has been modelled, information about orientation can be extracted from it, according to the following:

$$\mathbf{H_{ACC}} = \begin{bmatrix} \frac{\partial \mathbf{a_{IMU}}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{z}} + \frac{\partial \mathbf{a_{IMU}}}{\partial \dot{\mathbf{q}}} \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{z}} & \frac{\partial \mathbf{a_{IMU}}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \dot{\mathbf{z}}} + \frac{\partial \mathbf{a_{IMU}}}{\partial \dot{\mathbf{q}}} \frac{\partial \dot{\mathbf{q}}}{\partial \dot{\mathbf{z}}} \end{bmatrix} \tag{114}$$

where, again, $\frac{\partial \mathbf{q}}{\partial \dot{\mathbf{z}}}$ is zero. Moreover, again **T** matrix is constant, so it can be excluded from the differentiation. Evaluating the partial derivatives with respect to position, the result is:

$$\begin{aligned}
\frac{\partial a_{loc_x}}{\partial v_{1_x}} &= \ddot{x} + x_{IMU}\ddot{v}_{1_x} + y_{IMU}\ddot{v}_{2_x} + z_{IMU}\ddot{v}_{3_x} \\
\frac{\partial a_{loc_x}}{\partial v_{1_y}} &= \ddot{y} + x_{IMU}\ddot{v}_{1_y} + y_{IMU}\ddot{v}_{2_y} + z_{IMU}\ddot{v}_{3_y} \\
\frac{\partial a_{loc_x}}{\partial v_{1_z}} &= \ddot{z} + x_{IMU}\ddot{v}_{1_z} + y_{IMU}\ddot{v}_{2_z} + z_{IMU}\ddot{v}_{3_z} + g \\
\frac{\partial a_{loc_y}}{\partial v_{2_x}} &= \frac{\partial a_{loc_x}}{\partial v_{1_x}} = \frac{\partial a_{loc_z}}{\partial v_{3_x}} \\
\frac{\partial a_{loc_y}}{\partial v_{2_y}} &= \frac{\partial a_{loc_x}}{\partial v_{1_y}} = \frac{\partial a_{loc_z}}{\partial v_{3_y}} \\
\frac{\partial a_{loc_y}}{\partial v_{2_z}} &= \frac{\partial a_{loc_x}}{\partial v_{1_z}} = \frac{\partial a_{loc_z}}{\partial v_{3_z}}
\end{aligned} \tag{115}$$

This leads to:

$$\mathbf{H}_{\mathbf{ACC}} = \begin{bmatrix} \frac{\partial \mathbf{a}_{\mathbf{IMU}}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{z}} & 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{a}_{\mathbf{IMU}}}{\partial \mathbf{q}} \mathbf{R} & 0 \end{bmatrix} \tag{116}$$

Naming, for notation shortness, the three terms in (115):

$$\begin{aligned}
\ddot{r} &:= \ddot{x} + x_{IMU}\ddot{v}_{1_x} + y_{IMU}\ddot{v}_{2_x} + z_{IMU}\ddot{v}_{3_x} \\
\ddot{s} &:= \ddot{y} + x_{IMU}\ddot{v}_{1_y} + y_{IMU}\ddot{v}_{2_y} + z_{IMU}\ddot{v}_{3_y} \\
\ddot{t} &:= \ddot{z} + x_{IMU}\ddot{v}_{1_z} + y_{IMU}\ddot{v}_{2_z} + z_{IMU}\ddot{v}_{3_z} + g
\end{aligned} \tag{117}$$

It results:

$$\frac{\partial \mathbf{a}_{\mathbf{loc}}}{\partial \mathbf{q}} = \begin{bmatrix} 0 & \dots & 0 & \ddot{r} & \ddot{s} & \ddot{t} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & \ddot{r} & \ddot{s} & \ddot{t} & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ddot{r} & \ddot{s} & \ddot{t} & 0 & \dots & 0 \end{bmatrix} \tag{118}$$

4.4 HALL-EFFECT SENSORS

Every vehicle wheel is equipped with a Hall effect sensor which works in a similar way to an encoder: given the brake disc with holes – see Fig. 31 – it counts the number of holes which have passed in front of itself during wheel rotation. Every metallic disc is mounted on the wheel shaft and has forty holes, so giving the resolution of 9°. Therefore, a direct – even though it is not so accurate – measurement of every wheel angle can be obtained. The consequence is that the observer output matrix has four *one* terms at the correct position. Given the wheel angle vector:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} \tag{119}$$

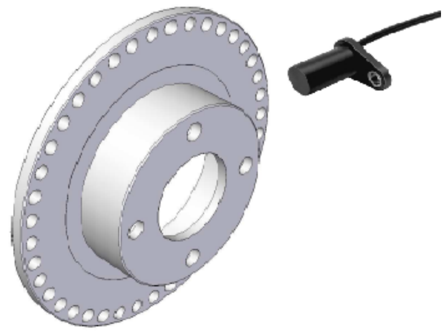

Figure 31: CAD model of a brake disk with Hall effect sensor.

This leads to:

$$\frac{\partial \boldsymbol{\theta}}{\partial \mathbf{z}} = \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix} \tag{120}$$

It is worth noticing that this part is linear, so a behaviour close to the optimum – like in position part – is expected.

4.5 CODE IMPLEMENTATION

Observer source was implemented in FORTRAN, into the *simbuggy* project[12]. It was divided into some *modules*, everyone of them containing a *subroutine*:

A. *observer init* is the initialization module, containing all the variables and vectors to be used later. In addition, all the operations which can be done before running the simulation – or which do not require recalculating some values at every time step – can be done off-line in the corresponding subroutine.

B. *compute gain* is a subroutine which performs gain calculation in scalar form, as it will be seen later.

C. *H calc* is the module containing the output matrix calculation and building. In this function the partial derivatives of gyroscopes and accelerometers are computed, since they are the ones which need to be calculated at every time-step.

D. *virtual meas* is the function which computes sensor models and provides virtual measurements to the observer.

12 This is the name of vehicle simulation software implemented by University of A Coruña LIM staff.

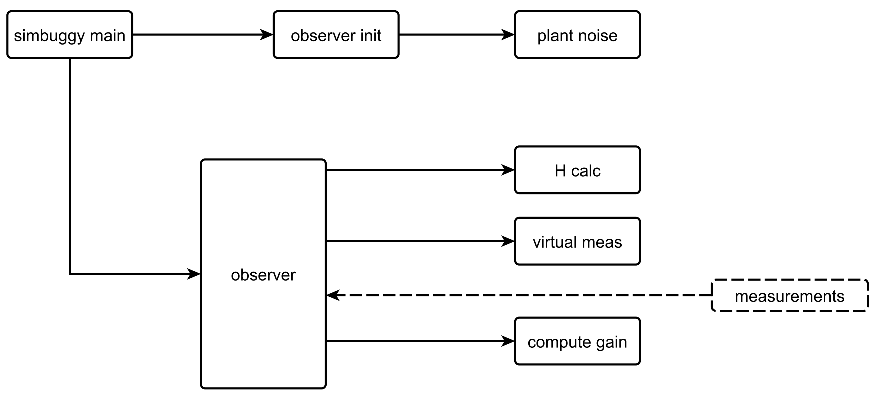

Figure 32: Calls hierarchy. External measurements are underlined with dashed line, in order to specify they are input for the observer.

E. *plant noise* is the function which calculates plant noise matrix, following *Van Loan's* method, as it will be seen later.

F. *observer* is the key function, the observer itself. It receives both the real and the virtual measurements, it has the gain calculated by the *compute gain* function and then performs the multi-body model correction.

The call hierarchy is represented in Fig. 32. In that scheme the fact that the measurements come from outside the function and are inputs for the observer is underlined with the use of the dashed line.

4.5.1 *Observer initialization*

In this module – *observer init* – the vectors used for making the calculation is allocated, in particular, in pseudo-code:

```
% Variables declaration
Declare:

% State dimension
state_dimension = 20

% Mesurements number
measurement_number = 16

% State vector
errors = empty vector of dimension: state_dimension

%Kalman gain vector
L = empty vector of dimension: state_dimension

% Innovation vector
innovation = empty vector of dimension:
    measurement_number
```

```
        .
        .
        .
        % da/dq accelerometers matrix
    dhdq_accelerometers = empty matrix of dimension: (3,
        state_dimension)

    % dOmega/dq gyroscopes matrix
    dhdq_gyroscopes = empty matrix of dimension: (3,
        state_dimension)

    % dOmega/dqp gyroscopes matrix
    dhdqp_gyroscopes = empty matrix of dimension: (3,
        state_dimension)
    .
    .
    .
```

These are the vectors, matrices and variables to be used in other functions. There are some which are declared and initialized too, which are the ones that will not be modified after:

```
% Variables declaration
Declare:

% F matrix
F = already filled matrix of dimension: (state_dimension,
     state_dimension)

% GPS drp/dq matrix
dhdq_GPS= already filled matrix of dimension: (3,
    state_dimension)
.
.
.
% Measurement noise
W = = already filled vector of dimension: state_dimension

% Plant noise
N = already filled vector of dimension:
    measurement_number      .
.
.
```

As showed in call hierarchy scheme of Fig. 32, the plant noise matrix is built by a specific function:

```
% Have plant noise matrix built
Q = plant_noise(N);
```

It is worth noticing that this function is executed *once and before the simulation*, so real-time efficiency is not required.

4.5.2 *Plant noise matrix calculation*

The plant noise matrix calculation is performed into the subroutine of the same name, and it uses *Van Loan's method* for having it computed in discrete-time [14, 34]. The procedure proposed in the cited text models the continuous time stochastic system – so formulating plant noise matrix $\mathbf{W}(t)$ – and then discretizes it in order to obtain $\mathbf{W_k}$, following the method of the matrix exponential formulated by Charles Van Loan.
Given the LTI continuous-time system:

$$\dot{\mathbf{x}}(t) = \mathbf{F}\mathbf{x}(t) + \mathbf{G}\mathbf{w}(t), \qquad \mathbf{w}(t) \in N(0, W_c) \tag{121}$$

where W_c is the plant noise covariance, plant noise is introduced in accelerations, since they are the main cause of uncertainties in multibody models. So, $\mathbf{G}$ will be the identity matrix, $\mathbf{w}(t)$ is a vector of zeros in its first half, and of noise coefficients in its second half. Acceleration noise will be, when discretised, a velocity difference noise. $\mathbf{M}$ matrix can now be defined:

$$\mathbf{M} := \begin{bmatrix} -\mathbf{F} & \mathbf{G}W_c\mathbf{G}^T \\ \mathbf{0} & \mathbf{F}^T \end{bmatrix} \tag{122}$$

So, computing matrix exponential, it results:

$$\exp(\mathbf{M}) = \begin{bmatrix} \mathbf{\Psi} & \mathbf{\Phi_k}^{-1}\mathbf{W_k} \\ \mathbf{0} & \mathbf{\Phi}_k^T \end{bmatrix} \tag{123}$$

where $\mathbf{W_k}$ is the object of the calculation, and it can be determined through the following procedure, in MATLAB code:

```
% Build M matrix
M = DeltaT*[-F,G*Wc*G';zeros(n),F'];

% Compute matrix exponential
N = expm(M);

% Extract Phik
Phik = N(n+1:2*n,n+1:2*n)';

% Wk calculation
Wk = Phik*N(1:n,n+1:2*n);
```

This operation is not necessary for measurement noise since, using the scalar gain calculation algorithm, only scalar values are used. This will be seen in the following paragraph

4.5.3 *Kalman gain calculation*

Kalman gain calculation and error vector updating operation are performed by the *compute gain* subroutine. The Kalman gain is computed in a scalar way [14]: since in most of the cases measurement noise matrix is diagonal, it is convenient to treat each element as scalar. This essentially for:

A. Computational efficiency;

B. Numerical accuracy.

The former is an advantage in terms of computation time: it can be demonstrated that, given n measurements, this method computational charge is $O(n)$, whereas following the classical vectorial approach it grows to $O(n^3)$. The latter is directly related to matrix inversion: by avoiding this operation in the covariance updating, the algorithm becomes more robust against round-off errors.
It proceeds taking the single rows and columns of $\mathbf{H}$ matrix: so, building it from the various parts – acceleration, angular velocity, GPS... – is no more required. Naming $\mathbf{H}^{[i]}{}_k$ the i-th row of $\mathbf{H}$ matrix at k time-step, and $\mathbf{L}^{[i]}{}_k$, z^i_k and N^i_k the i-th gain vector, measurement and noise coefficient respectively, it follows:

$$\begin{aligned}
\mathbf{L}^{[i]}{}_k &= \frac{1}{\mathbf{H}^{[i]}{}_k \mathbf{P}_{k|k-1} \mathbf{H}^{[i]}{}^T_k + N^i_k} \mathbf{P}_{k|k-1} \mathbf{H}^{[i]}{}_k \\
\mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{L}^{[i]}{}_k \mathbf{H}^{[i]}{}_k \mathbf{P}_{k|k-1} \\
\mathbf{x}_{k|k} &= \mathbf{x}_{k|k-1} + \mathbf{L}^{[i]}{}_k (z^i_k - \mathbf{H}^{[i]}{}_k \mathbf{x}_{k|k-1})
\end{aligned} \tag{124}$$

It can be noticed that in the first operation the denominator is scalar: so, matrix inversion has been effectively avoided. The Kalman gain, at every loop iteration, is a column, so the state estimate is updated by the quantity which is related to the i-th state variable. At the end of the loop, not only the calculation of Kalman gain has been carried out, but also the updating step has been completed.
It is worth noticing that, in this application, this way of calculating the Kalman gain is particularly convenient: in fact, since the filter faces a multi-rate sensors updating working condition, this procedure is suitable for calculating more or less gains, depending on measurement availability. If the classic algorithm was used, three – remember: the GPS can be available or not, and when it is, COG signal can be reliable or not – different measurement noise matrices were to be allocated, and a check over which one is to be used should be done at every time-step.
Moreover, this procedure allows adding or removing a sensor without changing matrix dimension and loop indices: this operation can be done simply adding a flag to each sensor, indicating its availability. This can be very useful for debugging operations: removing a single sensor without deleting the parts of the matrices which are related to it means to work *without state updating* but *with covariance one*: so, the state will not be updated, but filter divergence will be avoided.

4.5.4 *Model correction*

Observer function, in addition to calling all the other functions, performs model correction: starting from the updated state vector which is returned by *compute gain* function, the estimation error on all the natural coordinates can be computed. That is, all the errors in positions and velocities can be determined, so the model can be corrected simply operating on vectors $\mathbf{q}$ and $\dot{\mathbf{q}}$, adding or subtracting the errors. In order to correct position vector, although the errorEKF formulated in [30] solves kinematic velocity problem for doing it, the position algorithm has been used. In fact, using velocity one, the iterative Newton-Raphson calculation would have been avoided, but less accurate corrections would have been done. So, integrator convergence – *which works iteratively, too* – would have taken much more time. This, together with the low accuracy of corrections, is a disadvantage which makes preferable the resolution of kinematic position problem. So, the correction is performed adding the errors over the independent coordinates to vectors $\mathbf{q}$ and $\dot{\mathbf{q}}$ at the correct positions, then solving the kinematic problems of position and velocity, since the added quantities are degrees of freedom of the mechanism. Doing this, every natural coordinate is incremented of the correspondent error, so correcting the model. In pseudo-code:

```
%Adding position DOF errors to q
q = q + position errors

% Position correction
q = solve position kinematic problem

% Adding velocities DOF to qp
qp = qp + velocity errors

% Velocity correction
qp = solve velocity kinematic problem
```

It is worth noticing, however, that this is not a good correction for a multi-body model: in fact, to correct velocity in this way provokes a discontinuity, which would require an acceleration – and so force – impulse. Position correction is even more absurd: it is equivalent to *teleport*.
These are ways of correcting which are badly physically acceptable: even though it is a simulation model, the multi-body formulation could not be capable of converging in real-time if big errors were present, since integrator iteration number would become higher.
For this reason, an observer modification was thought: if acceleration errors were added to filter state, after their estimation, through inverse dynamics problem solution, force errors could be estimated too. This would lead to force correction, which are the main unknown in-

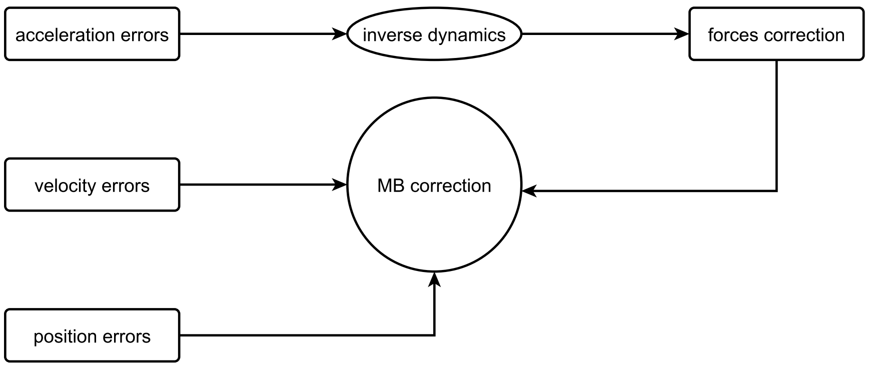

Figure 33: Observer extension through force errors estimation and correction.

puts to the system. Moving from the main dynamic formulation in (14) the problem to be solved is the following:

$$\begin{aligned} \mathbf{R}^T\mathbf{M}\mathbf{R}\ddot{\mathbf{z}} &= \mathbf{R}^T(\mathbf{Q}-\mathbf{M}\mathbf{S}\mathbf{c}) \\ \mathbf{c} &= -\dot{\mathbf{\Phi}}_\mathbf{q}\dot{\mathbf{q}} \end{aligned} \tag{125}$$

After some calculation, it results:

$$\mathbf{R}^T\mathbf{M}\mathbf{R}\ddot{\mathbf{z}} + \mathbf{R}^T\mathbf{M}\dot{\mathbf{R}}\dot{\mathbf{z}} = \mathbf{R}^T\mathbf{Q} \tag{126}$$

this leading to **Q** determination. Once it has been find out, it has to be added to multi-body system generalized force vector, so correcting that model non-knowledge.
If the entire model correction was to be performed in this way, possibly a slower filter would be obtained in terms of bandwidth. Imagining that the force correction at k time-step could make the model entirely corrected at $k+3$, at $k+1$ a new set of corrected forces would be calculated, but this calculation would be quite bad, since the velocities are have not been corrected yet. Because of this, the improvement in multi-body dynamics rapidity and robustness would be possibly overshadowed.
If it was performed putting together force, velocity and position correction, the problem of the non-physically acceptability explained upon would be, however, less heavy. So, a trade-off between physical correctness and estimation performances has to be searched. A scheme of this is shown in Fig. 33.
Anyway, to correct the forces would solve the constant force error problem discussed in GPS paragraph: even if there was front wind, so producing a constant error in x direction, by correcting the force vector this problem would be eliminated. So, a filter behaviour which is nearer the optimum one would be expected.
However, this part is still to be added, and it could a possible future improvement to the project.

5 SIMULATION RESULTS

5.1 SIMULATED MANOEUVRES

Observer performances have been tested firstly performing two kinds of manoeuvres in a virtual plane ground: a straight one and a circular one. The model has two drivers to be adjusted in order to make it run: the steering angle and the torque applied to the wheels. In both of them the former has been set through the assignment of a linear displacement value of the mechanism rack. The latter works as mechanical power input to the system.
Simulation procedure is the same used in the four-bar linkage mechanism simulation: firstly the multi-body model runs as it was the real mechanical system. It is utilized as the *ground truth*, without running the observer, recording natural coordinates and virtual sensor values. Then they are processed in MATLAB, adding a white noise with the variances specified in the previous chapter.
After that, a second simulation is run, introducing an error in the multi-body model – in order to represent its imperfection – and running the observer in parallel. This is expected to correct the multi-body model, which would behave differently with respect to the ground truth, and force it to follow the recorded performances.
The error introduced into the multi-body model is a difference in the *tyre friction coefficient*: in fact this is, generally, an unknown parameter, since it depends on several factors which are impossible to accurately predict when modelling the system, such as tyre wear or road conditions. Moreover, a friction coefficient error becomes, in performing dynamical calculations, a *force* error. It is a reasonable choice, since forces are the main cause of uncertainties in multi-body simulations.
A test list, with simulation parameters, is shown in Tab. 4.

Manoeuvre	Frict. error	Time [s]	Torque [Nm]	Steer displ. [m]
Straight	0.015	20	100	-0.004
Circular	0.015	20	100	0.01

Table 4: Simulation parameters.

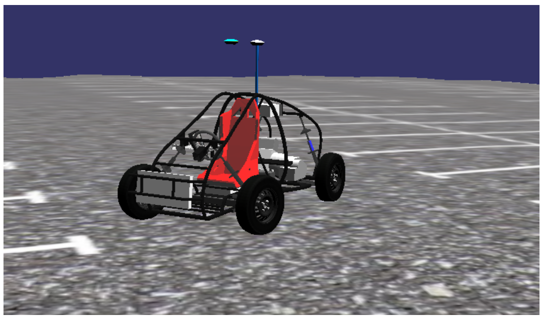

Figure 34: Vehicle simulation.

5.1.1 *Straight line manoeuvre*

Due to mechanical misalignments, especially of wheel hubs, steering rack displacement has to be set to a value different from zero, in order to move straight. After some attempts, a good value was found in 0.004 mm rightwards. The torque applied to each wheel was set to 100 Nm, in order to produce a quite relevant acceleration before reaching the velocity limit allowed by the tyre friction. Simulation time was set to 20 seconds.
No initial condition errors were taken into account, since it had been noticed that in the four bar linkage test they were corrected almost instantaneously. A difference of 0.015, between the ground truth and the multi-body model, in the tyre friction coefficient has been introduced. This value, resulted from some experimental tests, can be found also in literature [23].

It can be noticed that the multi-body model tends to diverge from ground truth behaviour, whereas the observer is capable of correcting the errors. In particular, plane Cartesian coordinates of positions and velocities – see Fig. 35-38 – are well estimated, and the errors remain in the neighbourhood of zero, whereas multi-body model errors diverge. In fact, observer presence makes maximum and mean squared error values decrease at least of one order of magnitude. The same thing happens to wheel angles and angular velocities, as it can be seen in Fig. 39-46.
In this case, although yaw error value is very small, observer makes performances worse. This is due to the fact that, since it performs a straight manoeuvre, multi-body model does not diverge in the yaw angle from the ground truth. On the other hand, observer works without a significant error to correct and preserves an offset, due to the misalignment between chassis and wheels and to the difference between course over ground and yaw angle. A similar thing happens to ω_z estimate: since the manoeuvre is straight, the multi-body model and the ground truth behave in a very similar way, making in both cases the angular velocity be almost zero. The results are shown in Fig. 47-48, whereas estimation error values are sum up in Tab. 5.

Coord.	max_{obs}	max_{mb}	rms_{obs}	rms_{mb}
x [m]	0.0338	30.210	0.0089	13.515
y [m]	0.0352	1.8367	0.0121	0.7488
$\dot{x}$ $[\frac{m}{s}]$	0.1547	3.0190	0.0451	1.7441
$\dot{y}$ $[\frac{m}{s}]$	0.1345	0.2297	0.0386	0.1147
θ_1 [rad]	0.0743	107.18	0.0224	47.937
θ_2 [rad]	0.1050	107.13	0.0223	47.913
θ_3 [rad]	0.0950	116.77	0.0263	52.222
θ_4 [rad]	0.1022	116.54	0.0258	52.119
$\dot{\theta}_1$ $[\frac{rad}{s}]$	1.1468	10.721	0.3103	6.1894
$\dot{\theta}_2$ $[\frac{rad}{s}]$	1.1907	10.716	0.3098	6.1863
$\dot{\theta}_3$ $[\frac{rad}{s}]$	1.6514	11.680	0.4266	6.7427
$\dot{\theta}_4$ $[\frac{rad}{s}]$	1.6200	11.656	0.4200	6.7293
yaw [rad]	0.0078	0.0011	0.0048	0.0004
ω_z $[\frac{rad}{s}]$	0.0140	0.0004	0.0051	0.0002

Table 5: Estimation errors in straight manoeuvre, committed by MB model with and without the observer respectively.

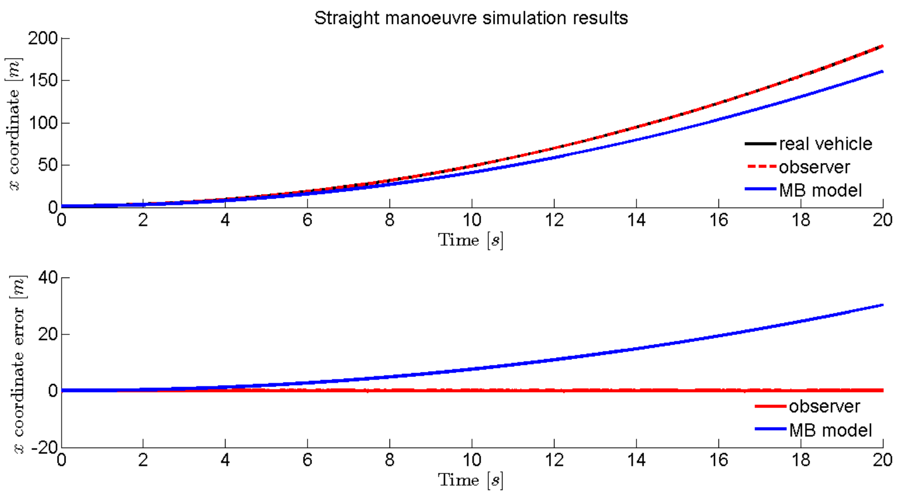

Figure 35: Estimated position of the vehicle in x-direction (top). Estimation error (bottom).

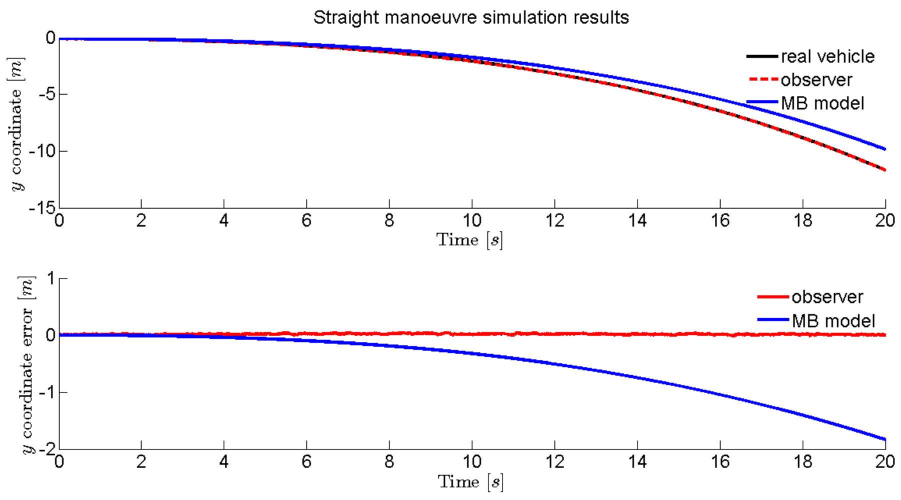

Figure 36: Estimated position of the vehicle in y-direction (top). Estimation error (bottom).

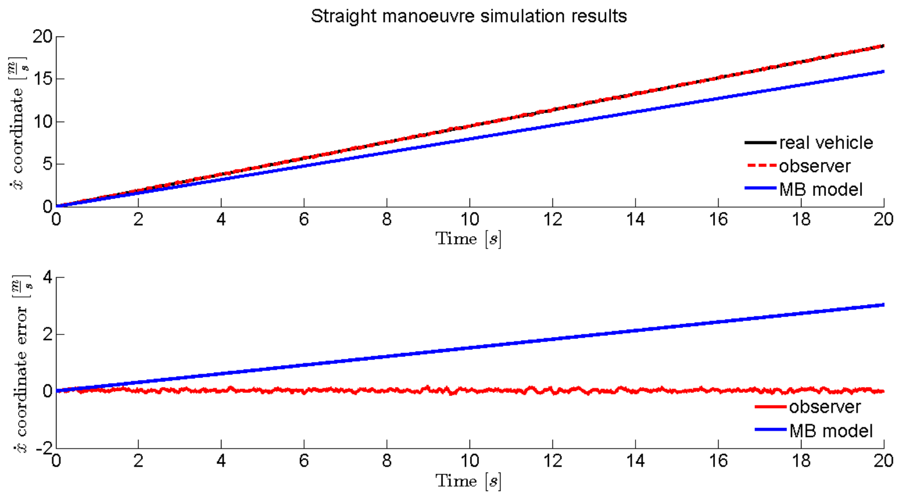

Figure 37: Estimated velocity of the vehicle in x-direction (top). Estimation error (bottom).

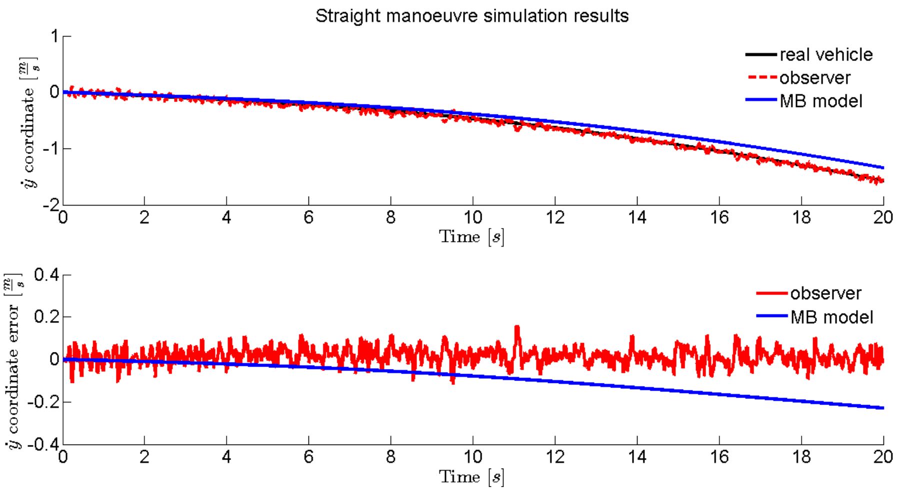

Figure 38: Estimated velocity of the vehicle in y-direction (top). Estimation error (bottom).

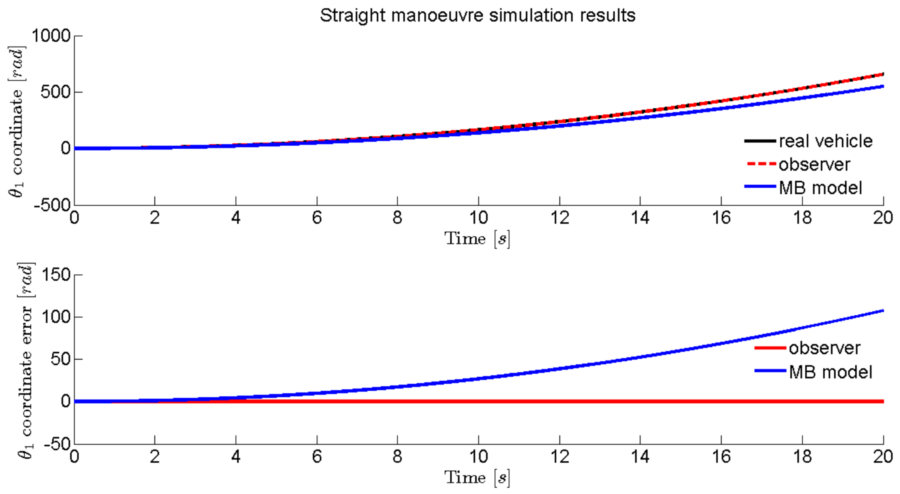

Figure 39: Estimated front right wheel angular position (top). Estimation error (bottom).

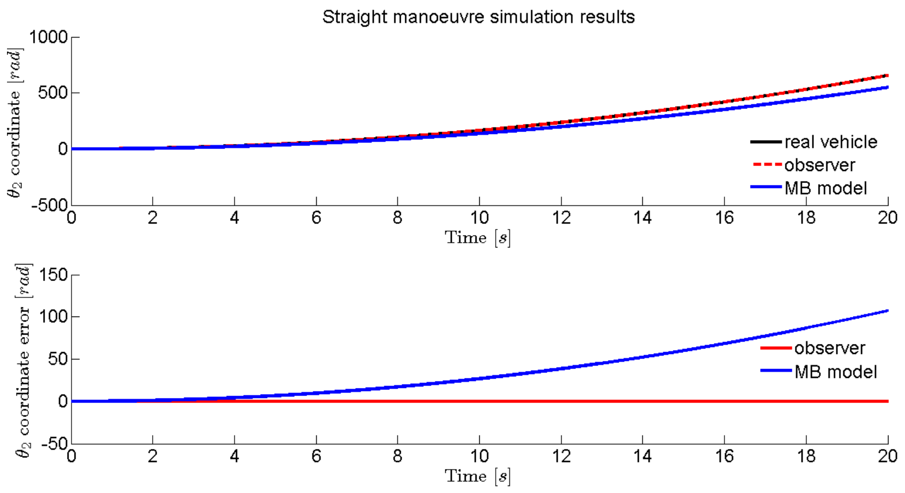

Figure 40: Estimated front left wheel angular position (top). Estimation error (bottom).

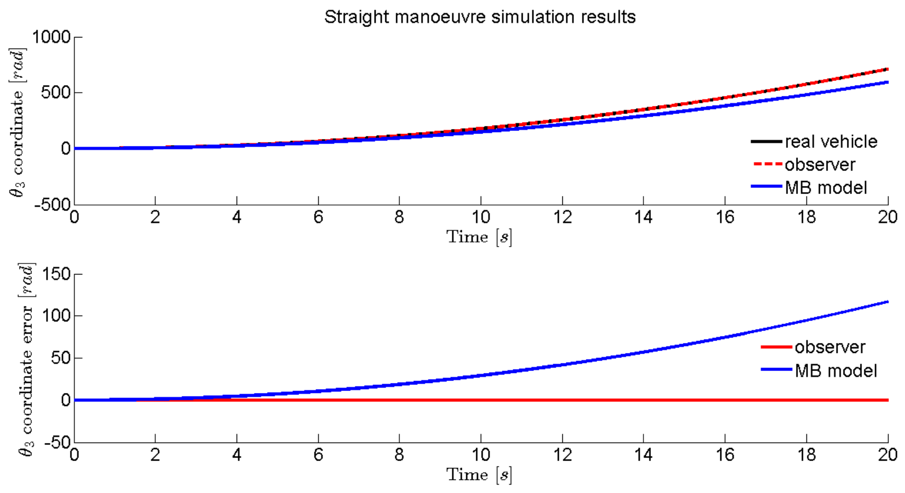

Figure 41: Estimated rear right wheel angular position (top). Estimation error (bottom).

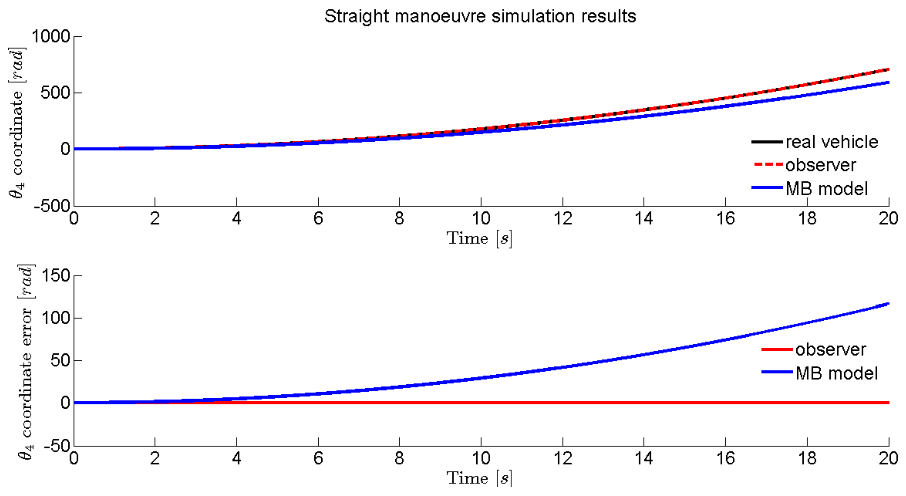

Figure 42: Estimated rear left wheel angular position (top). Estimation error (bottom).

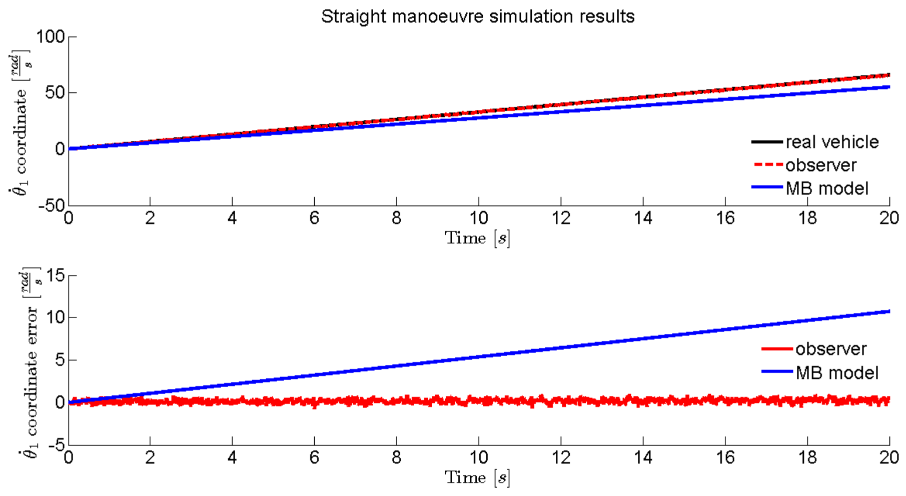

Figure 43: Estimated front right wheel angular velocity (top). Estimation error (bottom).

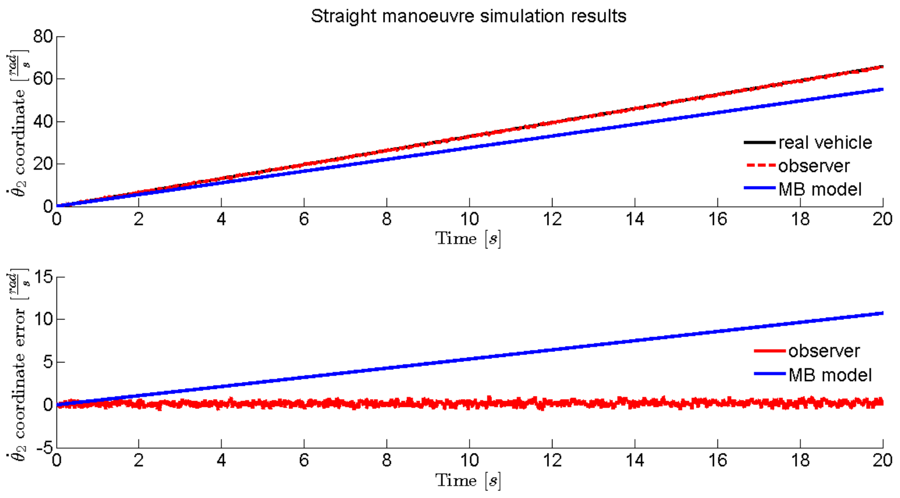

Figure 44: Estimated front left wheel angular velocity (top). Estimation error (bottom).

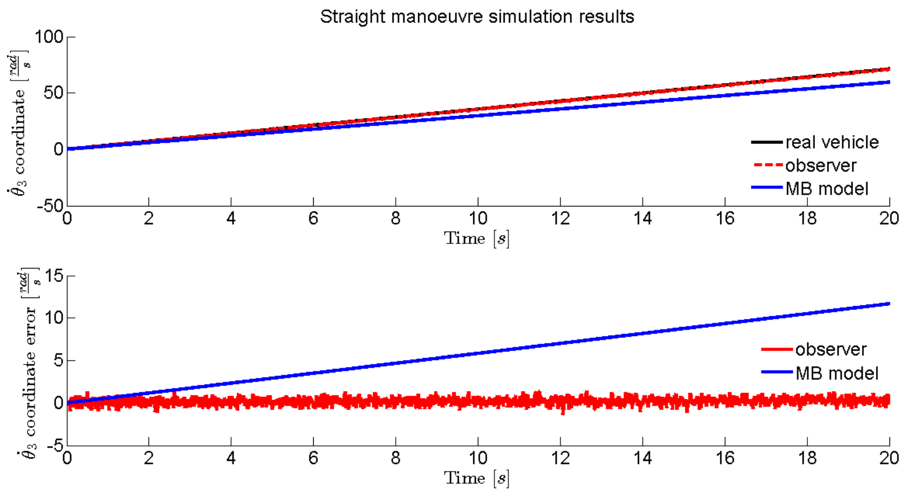

Figure 45: Estimated rear right wheel angular velocity (top). Estimation error (bottom).

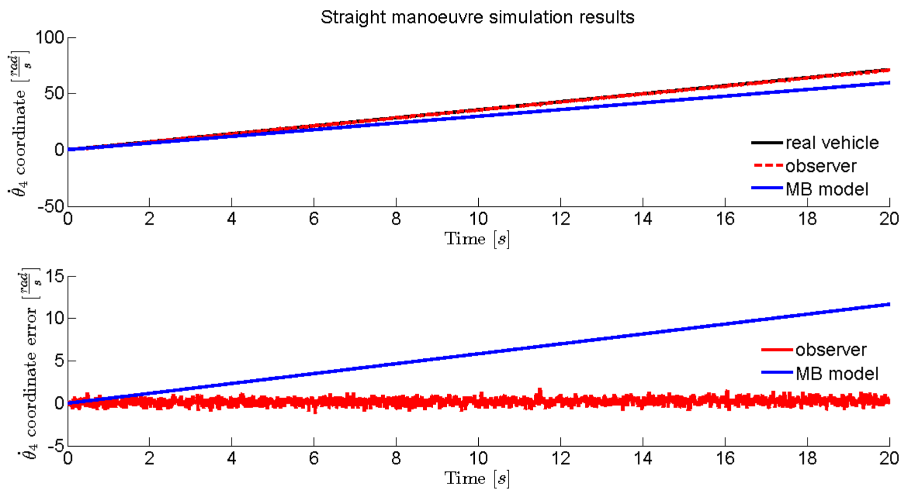

Figure 46: Estimated rear left wheel angular velocity (top). Estimation error (bottom).

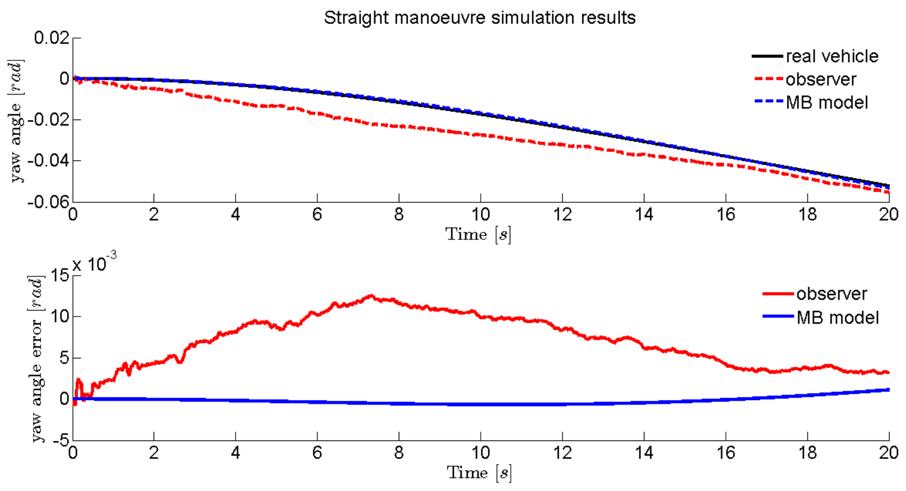

Figure 47: Estimated vehicle yaw angle (top). Estimation error (bottom).

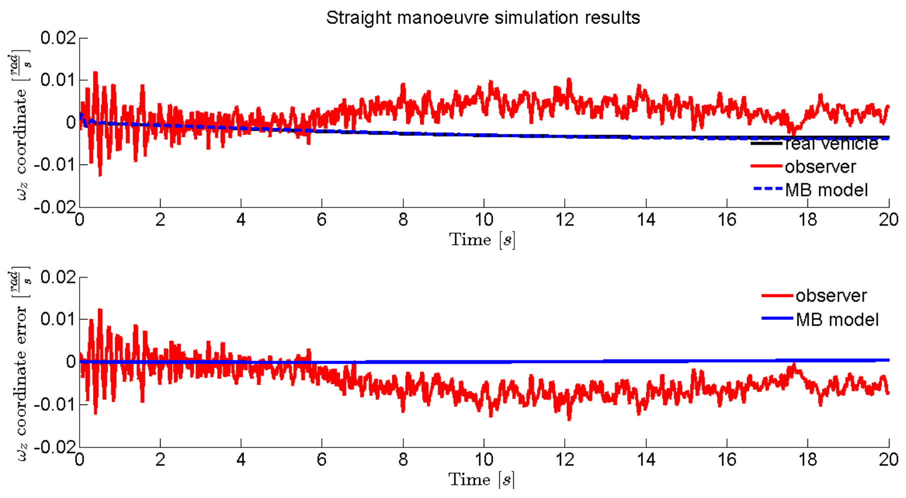

Figure 48: Estimated angular velocity of the vehicle about z-axis (top). Estimation error (bottom).

Coord.	$\max_{obs}$	$\max_{mb}$	rms_{obs}	rms_{mb}
x [m]	0.0398	11.752	0.0120	5.4598
y [m]	0.0374	10.106	0.0133	4.7388
$\dot{x}$ [$\frac{m}{s}$]	0.1624	4.9196	0.0448	2.2120
$\dot{y}$ [$\frac{m}{s}$]	0.1356	5.8118	0.0401	2.5558
θ_1 [rad]	0.0815	61.736	0.0221	33.238
θ_2 [rad]	0.0937	55.434	0.0236	29.868
θ_3 [rad]	0.0973	68.934	0.0254	36.929
θ_4 [rad]	0.0972	64.253	0.0250	34.196
$\dot{\theta}_1$ [$\frac{rad}{s}$]	1.0428	4.1167	0.2858	3.2786
$\dot{\theta}_2$ [$\frac{rad}{s}$]	0.9776	3.6802	0.2712	2.9411
$\dot{\theta}_3$ [$\frac{rad}{s}$]	2.8088	4.5946	0.4317	3.6657
$\dot{\theta}_4$ [$\frac{rad}{s}$]	3.8207	4.2657	0.4018	3.4207
yaw [rad]	0.0159	0.5889	0.0072	0.3608
ω_z [$\frac{rad}{s}$]	0.0143	0.0454	0.0042	0.0316

Table 6: Estimation errors in circular manoeuvre, committed by MB model with and without the observer respectively.

5.1.2 *Circular manoeuvre*

In order to move circularly, a steering rack displacement of 0.01 m leftwards was set configuring the simulation: the result is a counterclockwise turn. The torque applied to the wheels and the simulation time remained the same ones as in the previous test. In particular, this wheel torque value does not make them slip, yet it is sufficient to provoke a roll rotation of vehicle chassis. The error between the ground truth and the multi-body model is the same as in the previous test.

Also in this case, observer effectiveness can be noticed. In all the estimated variables it succeeds in correcting the multi-body model, making it follow the ground truth whereas the former diverges. In particular, also in this test observer presence makes the estimation error decrease several orders of magnitude in plane Cartesian coordinates and wheel angular ones, as it can be seen in Fig. 49-60. In this case, observer succeeds in correcting yaw angle error, although the small, already mentioned, offset remains. The angular velocity ω_z is corrected too, even though the error in multi-body model is small, so the correction effect is minor. Anyway, a one order of magnitude reduction in the RMS error value can be noticed: results are shown in Fig. 61-62. Estimation error values are sum up in Tab. 5.

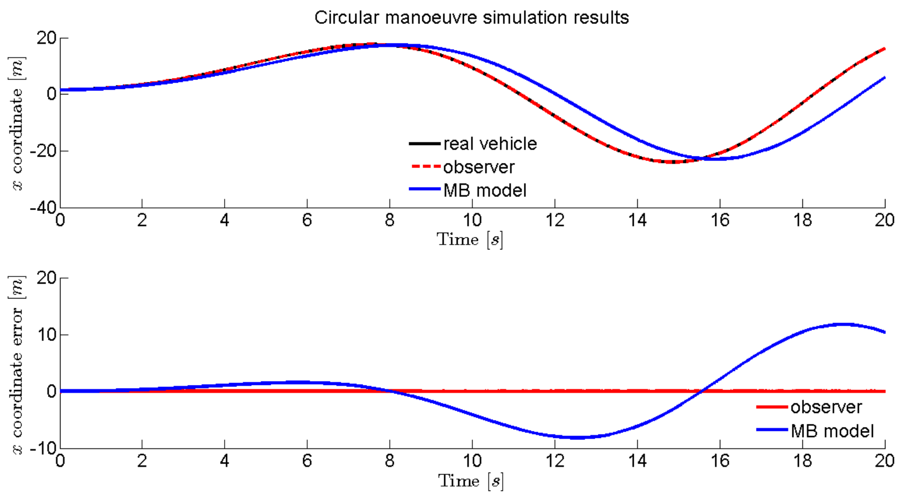

Figure 49: Estimated position of the vehicle in x-direction (top). Estimation error (bottom).

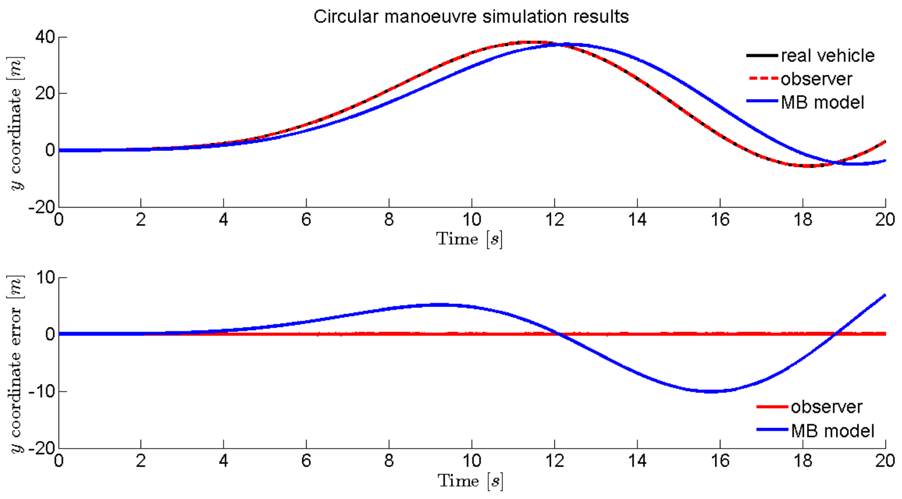

Figure 50: Estimated position of the vehicle in y-direction (top). Estimation error (bottom).

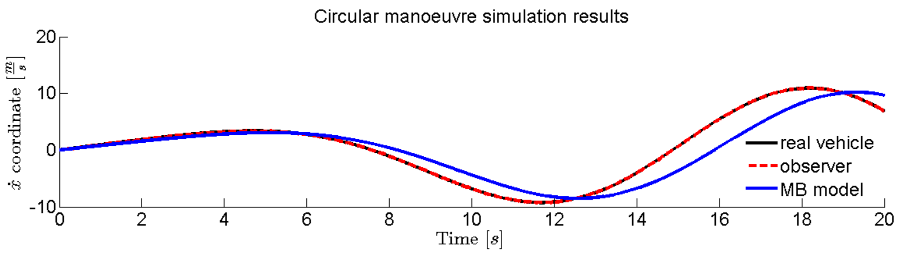
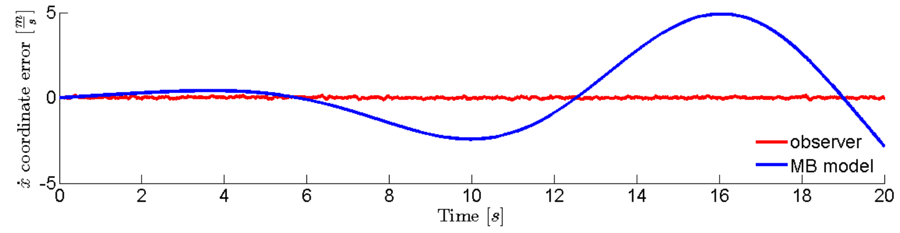

Figure 51: Estimated velocity of the vehicle in x-direction (top). Estimation error (bottom).

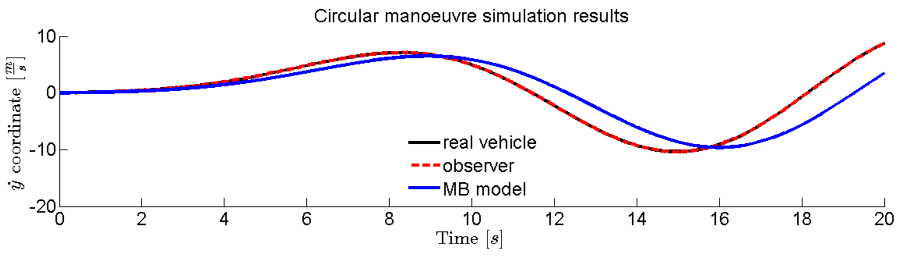
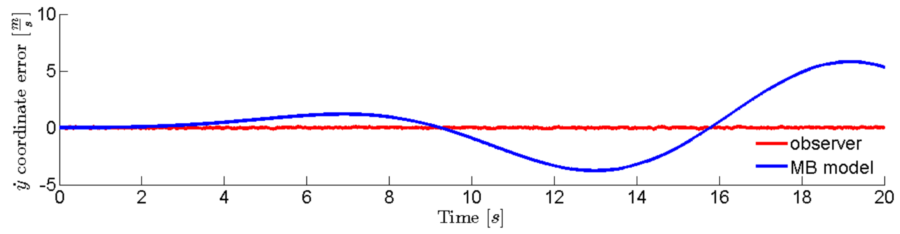

Figure 52: Estimated velocity of the vehicle in y-direction (top). Estimation error (bottom).

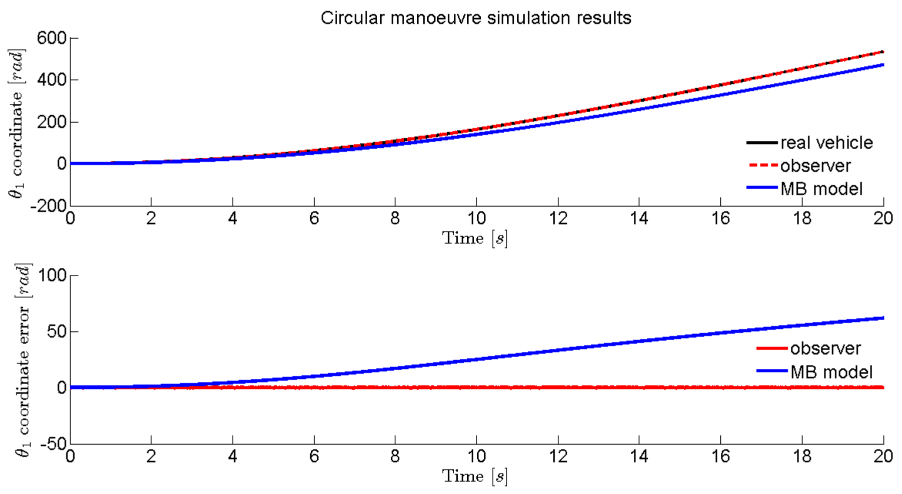

Figure 53: Estimated front right wheel angular position (top). Estimation error (bottom).

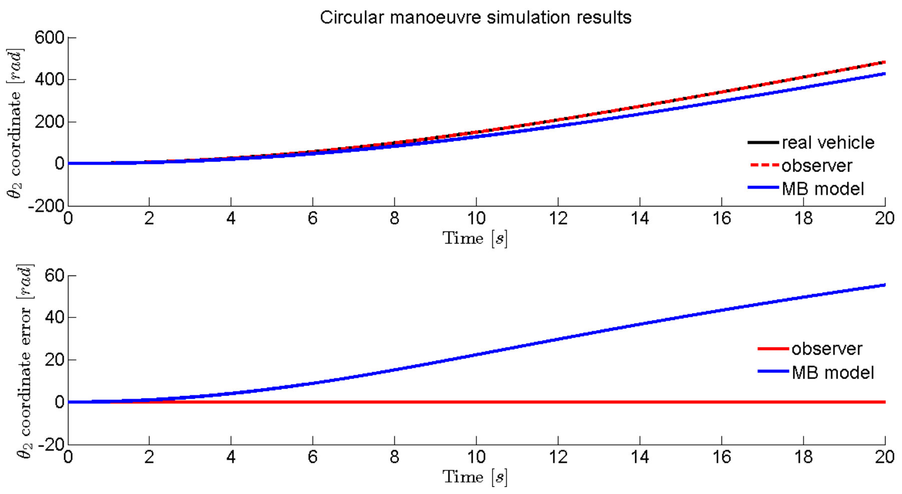

Figure 54: Estimated front left wheel angular position (top). Estimation error (bottom).

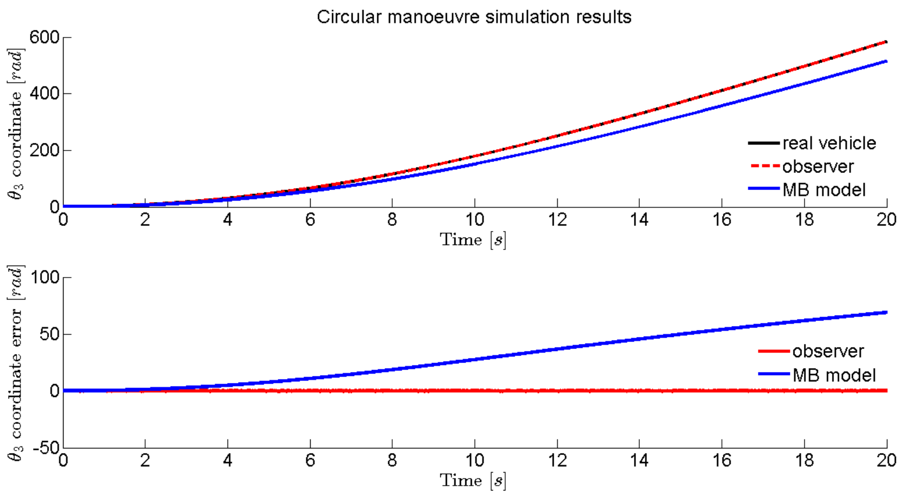

Figure 55: Estimated rear right wheel angular position (top). Estimation error (bottom).

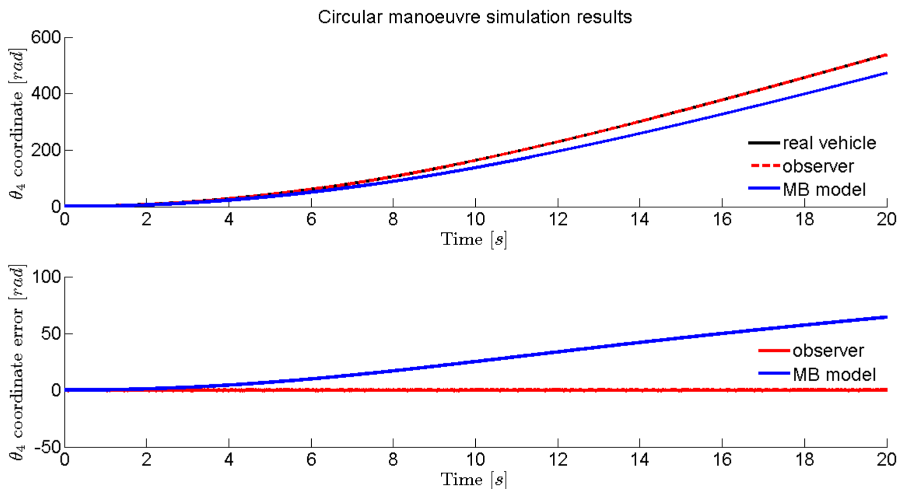

Figure 56: Estimated rear left wheel angular position (top). Estimation error (bottom).

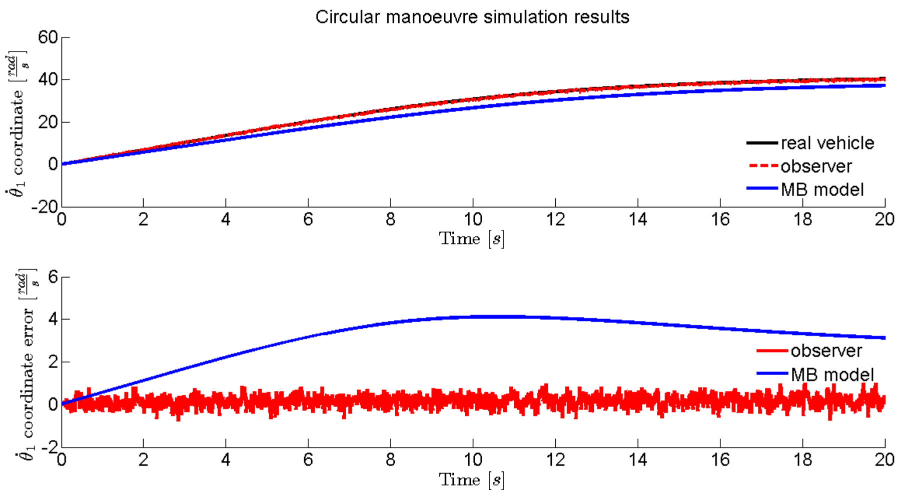

Figure 57: Estimated front right wheel angular velocity (top). Estimation error (bottom).

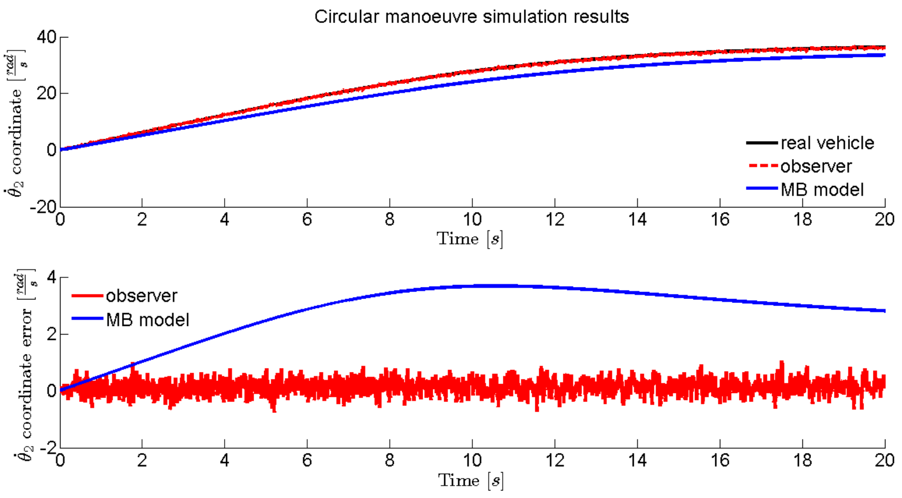

Figure 58: Estimated front left wheel angular velocity (top). Estimation error (bottom).

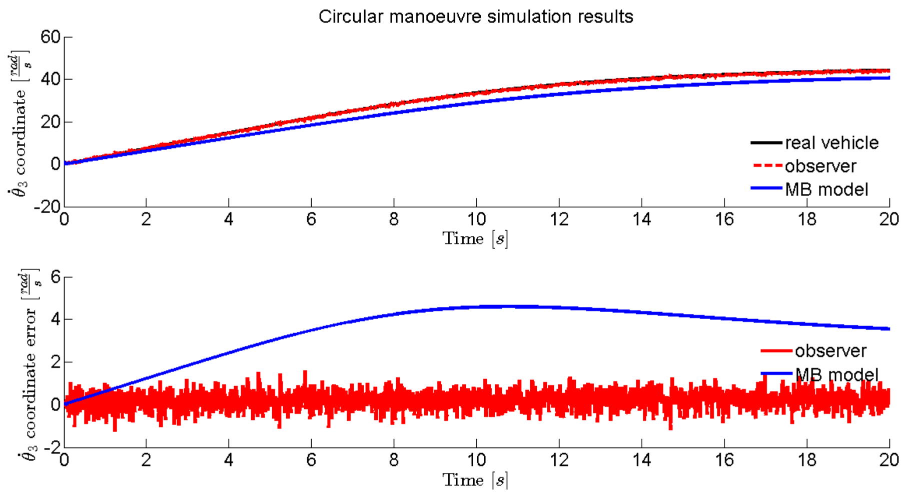

Figure 59: Estimated rear right wheel angular velocity (top). Estimation error (bottom).

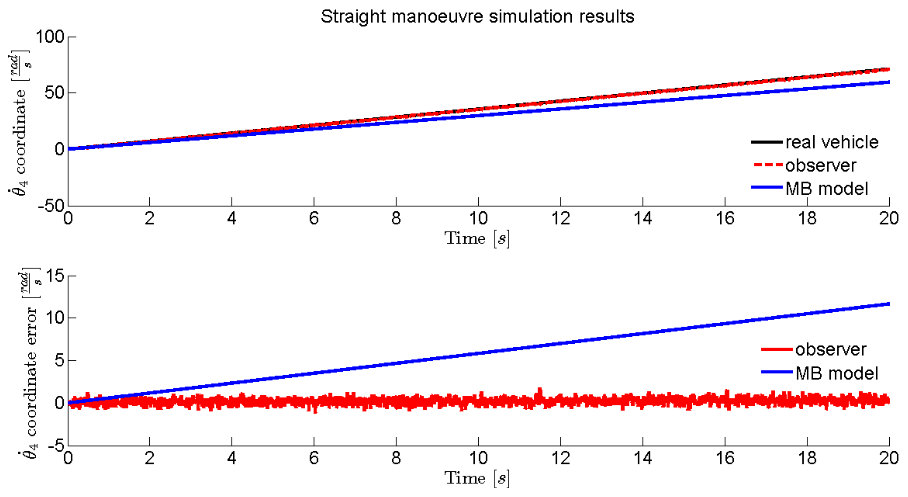

Figure 60: Estimated rear left wheel angular velocity (top). Estimation error (bottom).

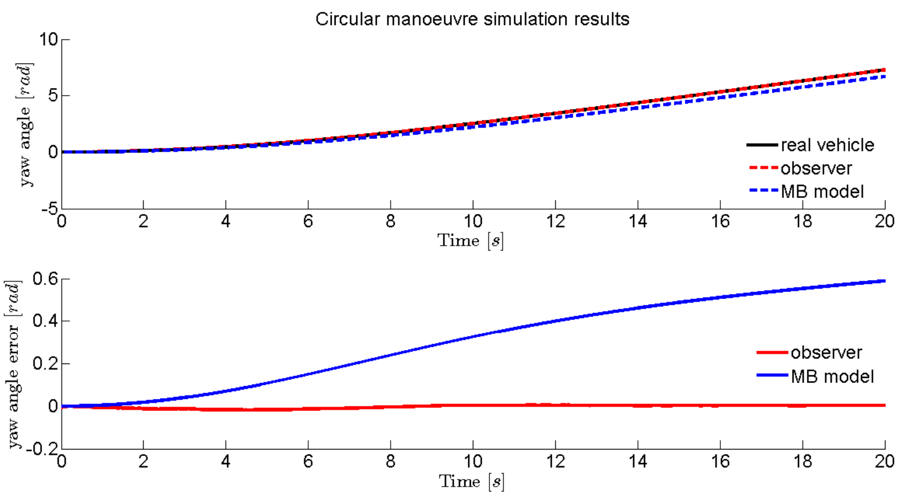

Figure 61: Estimated yaw angle of the vehicle(top). Estimation error (bottom).

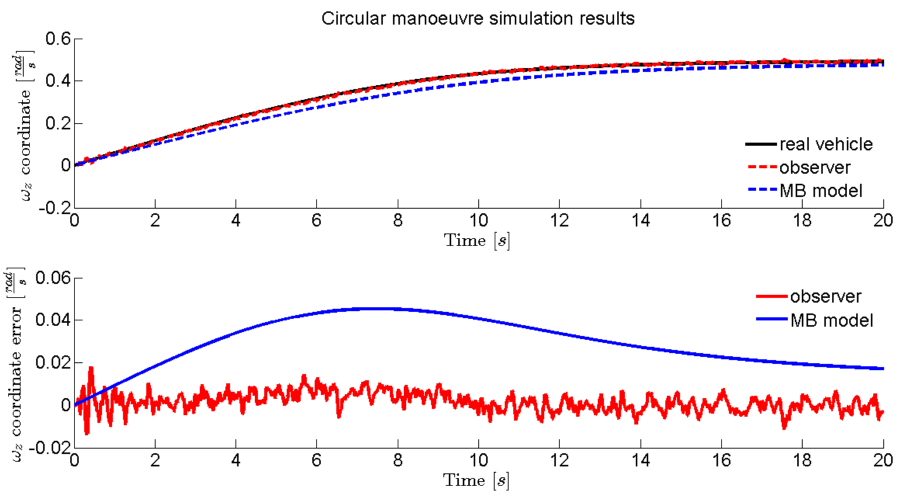

Figure 62: Estimated angular velocity of the vehicle about z-axis (top). Estimation error (bottom)

5.2 A MALATA PARKING MANOEUVRES

In addition to the previous tests, a more realistic one has been carried out. Instead of using the multi-body model as the ground truth, a set of real sensor data has been used. It is the set of buggy sensor data which was recorded during some manoeuvres done in Pabellón de A Malata[1] parking, when the model was validated. Wheel torque and steering rack displacement data are used as inputs to the multi-body model, other sensors are used to correct it through Kalman filtering.
The manoeuvre is circular, performing three clockwise turns. Parking ground had been modelled importing surveying data in a graphic software, so reproducing a real soil.

5.2.1 *Real sensors data manoeuvre*

In a first simulation, sensor data are used without any kind of preprocessing, since the signals are already affected by noise.
In spite of several factors that, working with real data, could make the results worse, it can be seen in Fig. 63-66 that Cartesian positions and velocities of the chassis are well estimated, whereas multi-body model behaviour is divergent. In particular, in this case too, observer makes estimation error decrease at least one order of magnitude. The same thing happens to wheel angular positions, as it is shown in Fig. 67-70. It can be noticed that the error in rear wheel estimates is bigger than the one in front wheels: this is due to the fact that the accuracy of Hall sensor measurements conveys engine vibrations, which affects holes count[2].
Yaw angle appears to be well estimated too, even though the available signal is a *course over ground* one: the plot it used only to give an idea of vehicle attitude correct estimation. The presence of the COG signal is the reason why the estimation is sensible only when the vehicle is moving, as it can be noticed looking at Fig. 71. Angular velocity ω_z – see Fig. 72 – estimate is quite good, too: the error is reduced about 50% in maximum value and about 60% in RMS one. Estimation error values are shown in Tab. 7.

5.2.2 *Manoeuvre with noise added to GPS position signal*

Buggy GPS is very accurate, since it has differential corrections in position: it can reach 2 centimetres accuracy. This is not reasonable for a commercial available car: due to this, another simulation was made, in order to reproduce more realistic conditions.

1 Estrada Malata, s/n, 15405 Ferrol, A Coruña. Latitude - Longitude 43.493992-8.24706.
2 It happens that, even if the vehicle is not moving, it appears like a wheel were rotating. This is due to the fact that, if a hole is near the *zero notch*, engine vibrations make it move about that notch, so that counted holes number increases.

Coord.	$\mathrm{max}_{\mathrm{obs}}$	$\mathrm{max}_{\mathrm{mb}}$	$\mathrm{rms}_{\mathrm{obs}}$	$\mathrm{rms}_{\mathrm{mb}}$
x [m]	0.1191	18.915	0.0080	3.6509
y [m]	0.0987	18.151	0.0081	3.4061
$\dot{x}$ $[\frac{m}{s}]$	0.8319	9.6530	0.0725	1.6717
$\dot{y}$ $[\frac{m}{s}]$	0.8664	10.348	0.0799	1.6616
θ_1 [rad]	0.2840	63.665	0.0814	32.420
θ_2 [rad]	0.2561	150.87	0.0621	108.50
θ_3 [rad]	0.9354	49.293	0.2061	26.295
θ_4 [rad]	0.7815	190.63	0.1827	135.50
yaw [rad]	0.2482	1.9863	0.0469	0.5954
ω_z $[\frac{rad}{s}]$	0.1212	0.2041	0.0253	0.0663

Table 7: Estimation errors in A Malata manoeuvre, committed by MB model with and without the observer respectively.

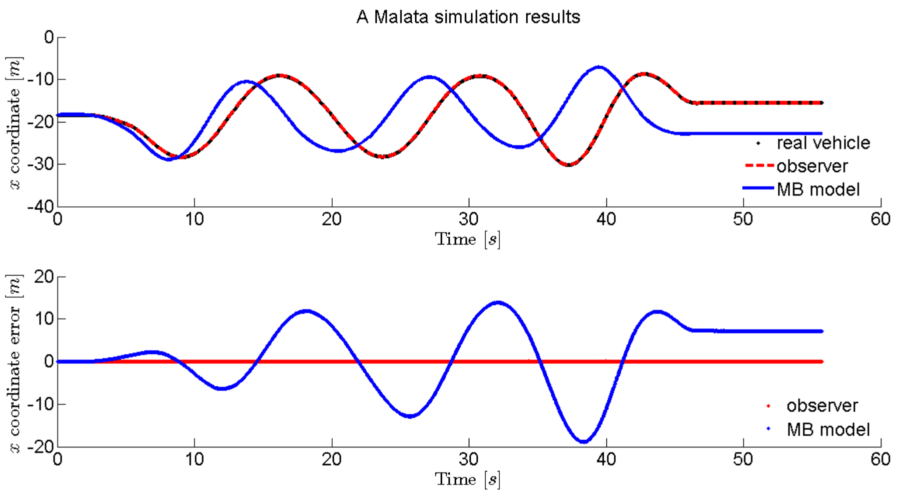

Figure 63: Estimated position of the vehicle in x-direction (top). Estimation error (bottom).

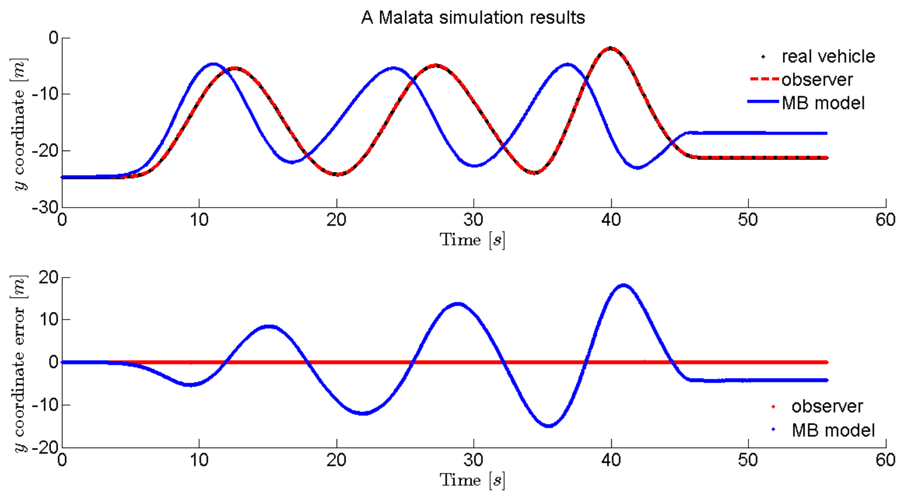

Figure 64: Estimated position of the vehicle in y-direction (top). Estimation error (bottom).

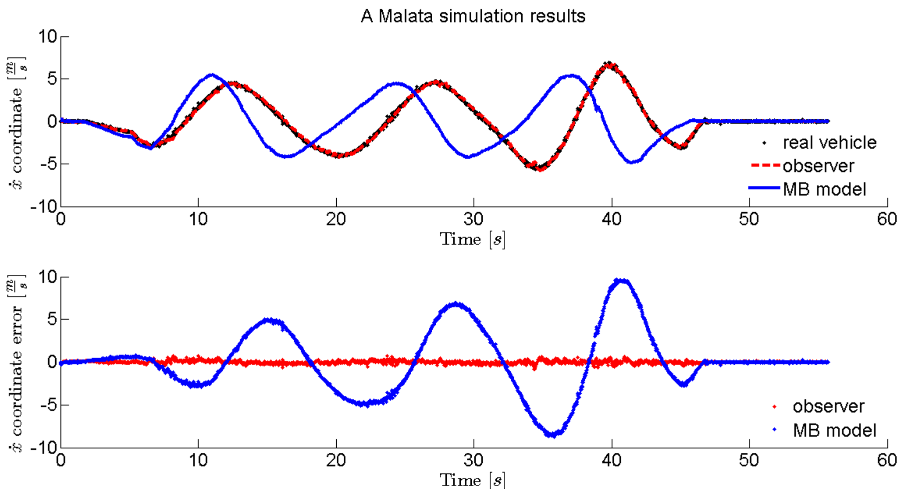

Figure 65: Estimated velocity of the vehicle in x-direction (top). Estimation error (bottom).

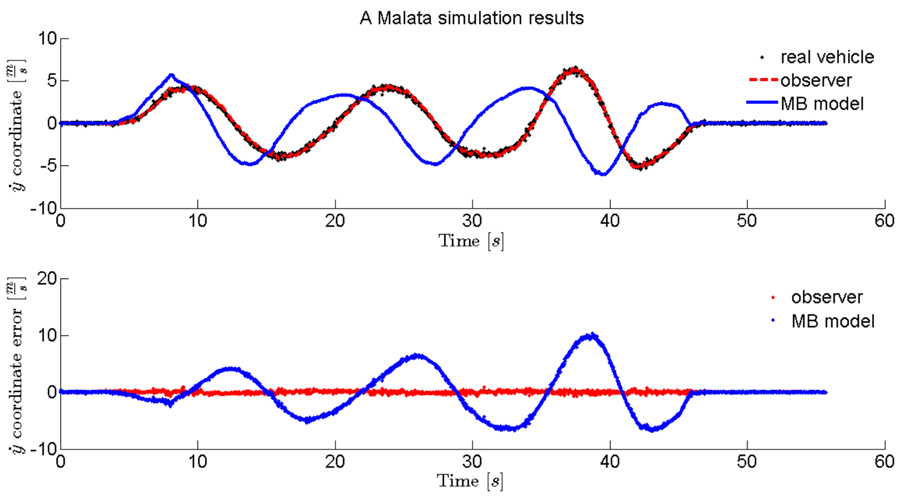

Figure 66: Estimated velocity of the vehicle in y-direction (top). Estimation error (bottom).

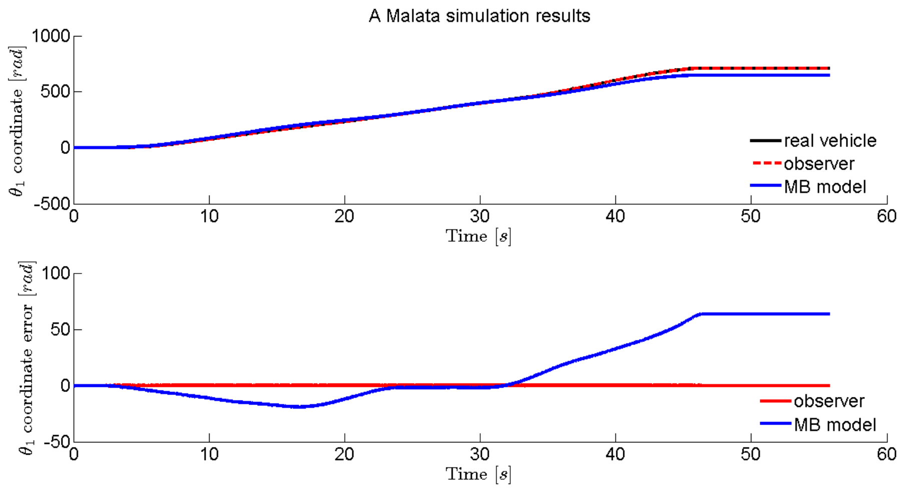

Figure 67: Estimated front right wheel angular position (top). Estimation error (bottom).

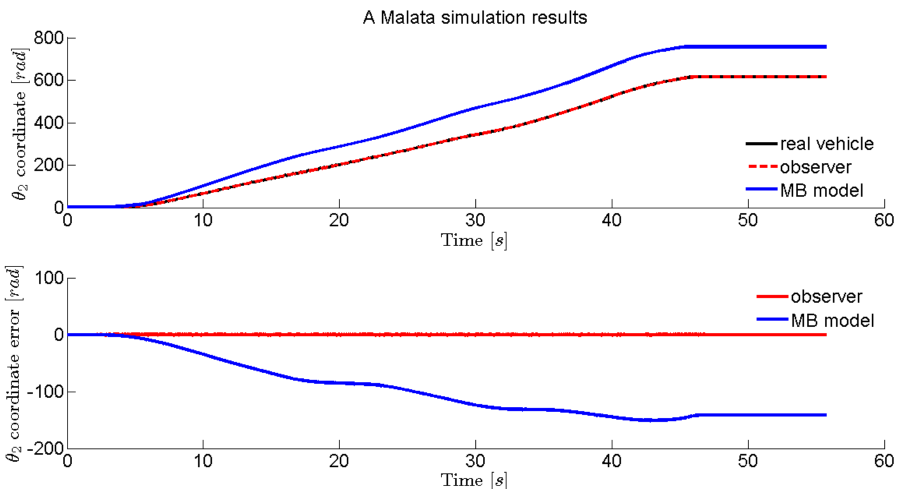

Figure 68: Estimated front left wheel angular position (top). Estimation error (bottom).

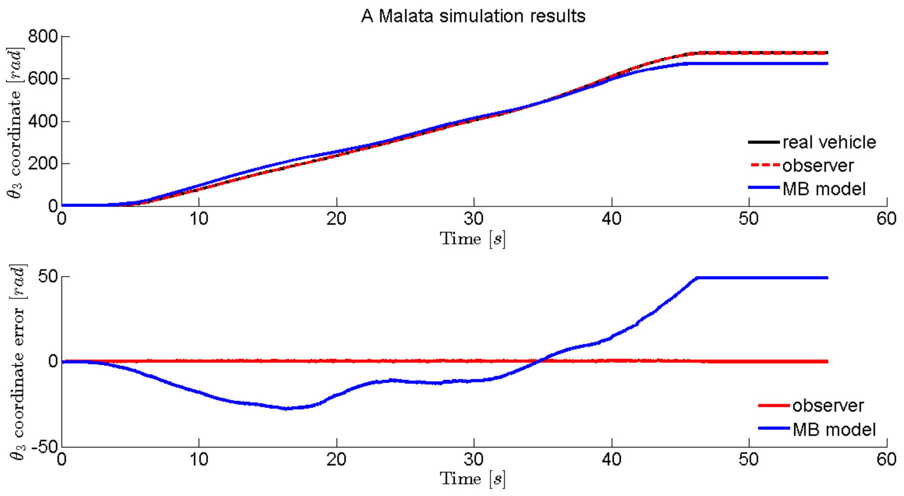

Figure 69: Estimated rear right wheel angular position (top). Estimation error (bottom).

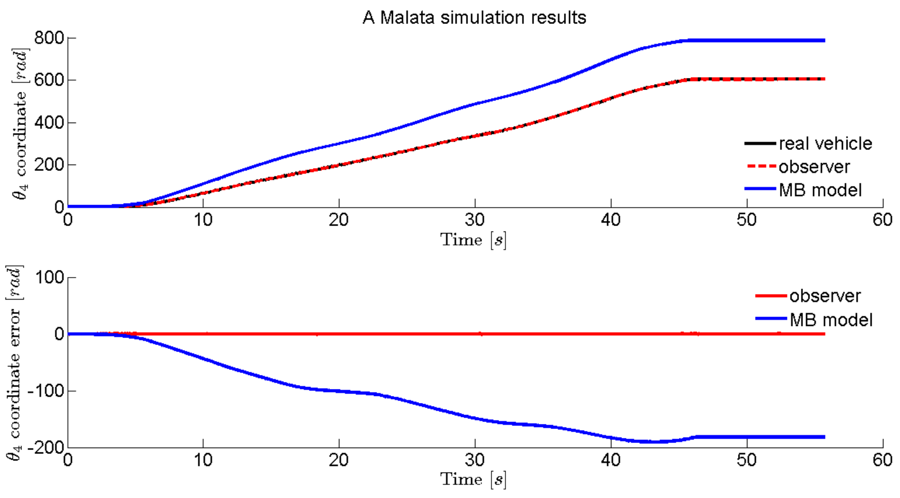

Figure 70: Estimated rear left wheel angular position (top). Estimation error (bottom).

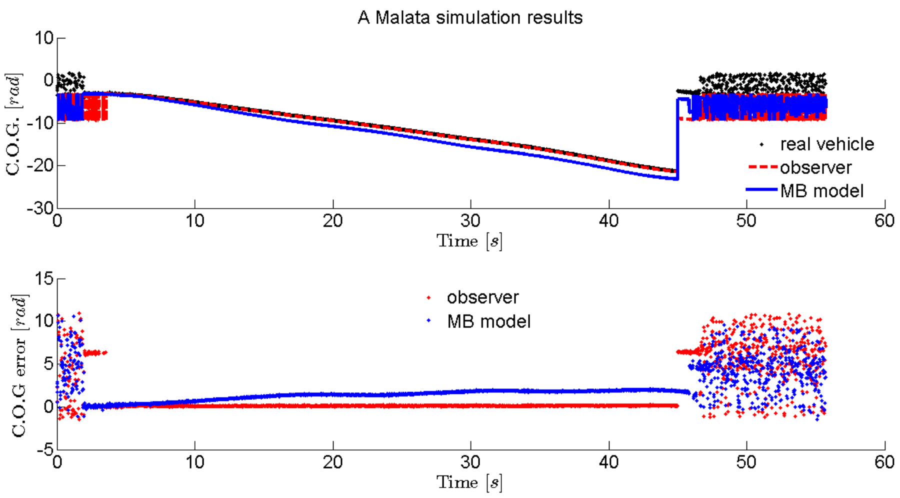

Figure 71: Estimated vehicle yaw angle, red and blue, versus course over ground, black (top). Estimation error (bottom).

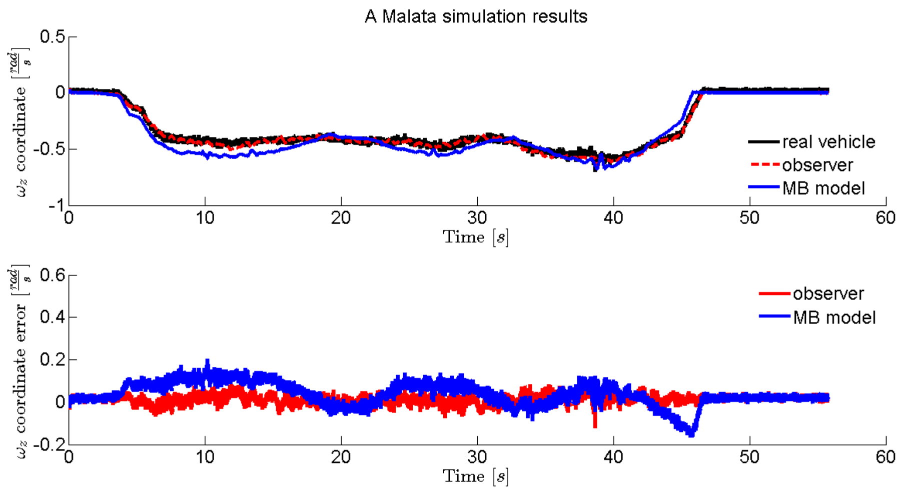

Figure 72: Estimated angular velocity of the vehicle about z-axis (top). Estimation error (bottom).

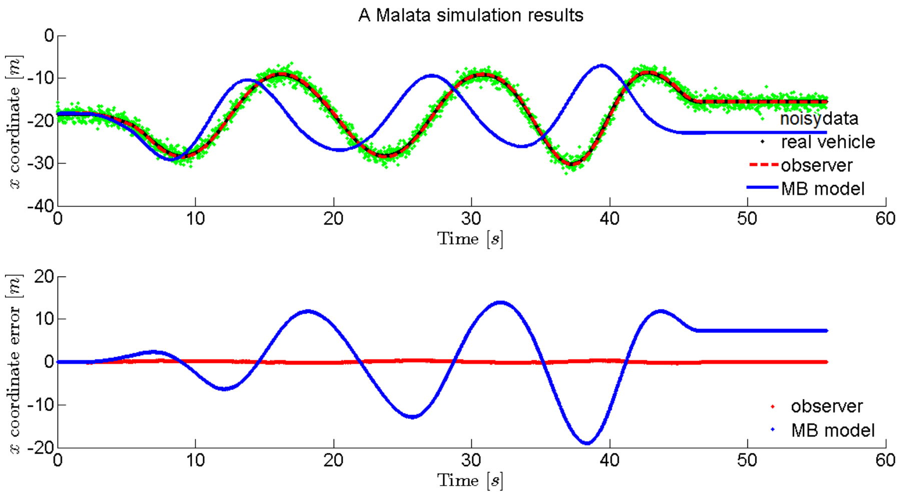

Figure 73: Estimated velocity in x-direction (top) with added GPS noise. Estimation error (bottom).

Position signal of GPS was processed before starting the simulation, adding a white noise of 1 metre variance: it should be more similar to one which can be bought with a normal car. Velocity – and so course over ground – signals were not modified, since differential corrections are related to position only.
As it can be seen in Fig.73-76, the observer goes on correcting the multibody model and avoiding its divergence, even though estimation accuracy is obviously worse. It is worth pointing out that the added noise affects only plane Cartesian coordinate estimates: wheel part is not affected by GPS signal corruption. This is the reason why only the plots related to these are shown. A small difference in yaw angle estimate can be seen with respect to the previous test, but it is quite negligible. Estimation error values are shown in Tab. 8.

Coord.	max_{obs}	max_{mb}	rms_{obs}	rms_{mb}
x [m]	0.4301	18.915	0.0706	3.6509
y [m]	0.4364	18.151	0.0759	3.4177
$\dot{x}$ $[\frac{m}{s}]$	1.1579	9.6530	0.1142	1.6717
$\dot{y}$ $[\frac{m}{s}]$	1.1850	10.348	0.1247	1.6616
θ_1 [rad]	0.2814	63.665	0.0780	32.420
θ_2 [rad]	0.2671	150.87	0.0632	108.50
θ_3 [rad]	0.9863	49.293	0.1922	26.295
θ_4 [rad]	0.7102	190.63	0.1930	135.50
yaw [rad]	0.2729	1.9863	0.0332	0.5954
ω_z $[\frac{rad}{s}]$	0.1177	0.2041	0.0236	0.0663

Table 8: Estimation errors in A Malata manoeuvre, committed by MB model with and without the observer respectively, with added GPS signal noise.

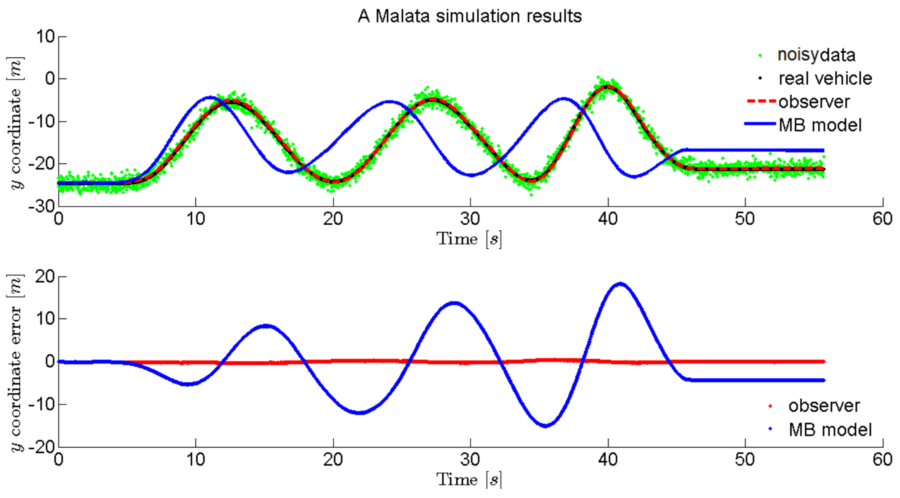

Figure 74: Estimated position in y-direction (top) with added GPS noise. Estimation error (bottom).

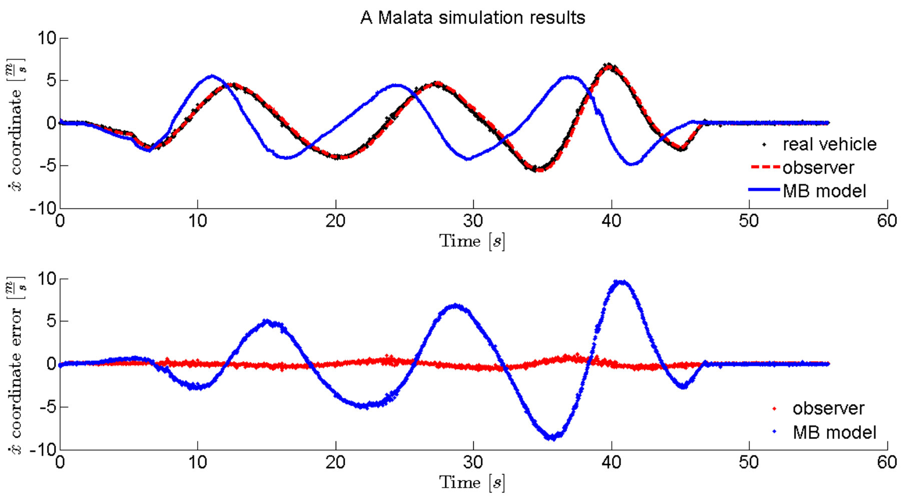

Figure 75: Estimated velocity in x-direction (top) with added GPS noise. Estimation error (bottom).

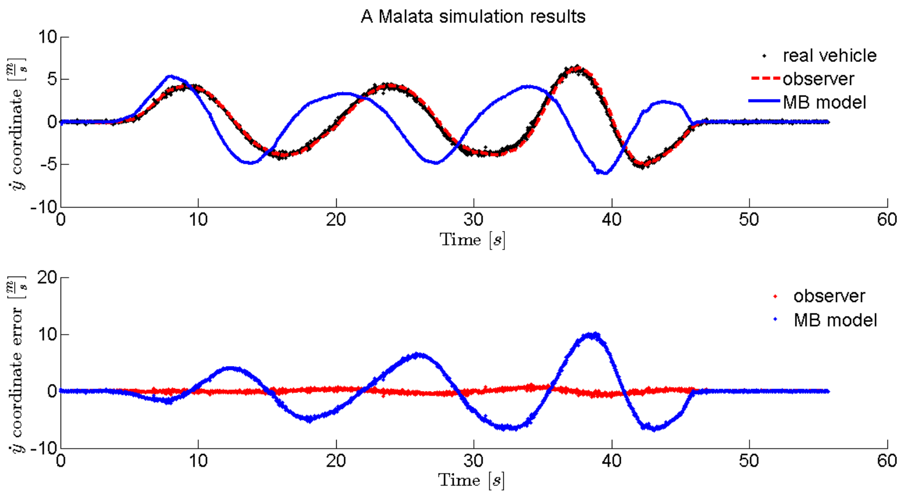

Figure 76: Estimated velocity in y-direction (top) with added GPS noise. Estimation error (bottom).

CONCLUSIONS

State observers are very useful tools, which allow estimating physical quantities of real systems otherwise unmeasurable. The aim of this work is to design observers capable to run in *real-time* and to perform the correction of multi-body models, in order to use such corrected models as virtual sensors. To this end a multi-body oriented state observer has been designed and its realization for a *dune buggy* ground vehicle has been thoroughly discussed.
The proposed observer is based on a modified version of the indirect – also called *error-state* – Kalman filter. Basically, it estimates position and velocity errors committed by multi-body models, exploiting measurements from sensors commonly available on cars. The estimated errors are then used to fully correct the multi-body models, making them reliable.

Experimental results confirm that the aim was well reached. It was found out that a trade-off between estimation performances, in terms of filter bandwidth, and multi-body model integration has to be searched. In fact, if filter response was very fast, this could lead to dynamics solving problems. On the other hand, if the multi-body model good integration was preferred, estimation performances could decrease considerably.
The kind of filter used allowed implementing an observer which runs in *real-time*, performing satisfying results. It is worth noticing, however, that a force-error estimation and correction stage should be added, in order to correct the multi-body model in a more proper way. In fact, if motion equations were respected, the integration would be surely easier and faster. This is, however, a possible future development and further improvement.
The most evident drawback of this filter adoption is the necessity of more measurements than in the classical Kalman filter for making the system observable. In fact, a loss of observability was underlined. However, a higher speed than the one possible with other filters is obtained, this giving the possibility of a real-time running.
An advantage, with respect to a classical Kalman filter, is its easy of design: a multi-body model state-space formulation is not needed. This allows a more *general* design operation, since it does not depend on dynamic equations, but only on sensor ones. These are all consequences of the state-transition matrix substitution with an integrator one, which allows coupling with the external multi-body model and makes the errorEKF a filter suitable to be used with multi-body models.

APPENDIX

A FOUR-BAR LINKAGE MECHANISM MATLAB MULTI-BODY SIMULATION SOURCE CODE

A.1 MAIN FUNCTIONS

A.1.1 *Main function*

```
%
%   Four bar linkage mechanism in natural coordinates
%   with errorEKF-based state observer
%
%   Author: Edoardo


clear all
close all
clc

addpath('functions')

%% Kinematic analysis

% 1 DOF => 4 constraint equations.  Putting basic points on the
    rotoidal
% couplings and on a,b points, we have:
%
%   (x1-xa)^2 + (y1-ya)^2 = L1^2
%   (x2-x1)^2 + (y2-y1)^2 = L2^2
%   (xb-x2)^2 + (yb-y2)^2 = L3^2
%   (x1-xa) = L1*cos(phi) OR (y1-ya) = L1*sin(phi)
%   (it depends on the value of angle phi)

% Setting constant values
[l, m, m_f, x] = kinematic_init();

% DOF
phi = degtorad(60); % position [rad]
phi_dot = 1; % velocity [rad/s]
phi_2dot = 0.0001; % acceleration [rad/s^2]

% DOF fake model
phi_f = degtorad(90);
phi_dot_f = -10;
phi_2dot_f = 0.0001;

% First iteration values
```

```
x1 = 0;
y1 = 4;
x2 = 8;
y2 = 3;

% First iteration values (fake model)
x1_f = 0;
y1_f = 2;
x2_f = 8;
y2_f = 3;

% Free coordinates vector
q = [x1; y1; x2; y2; phi];

% Free coordinates vector (fake model)
q_f = [x1_f; y1_f; x2_f; y2_f; phi_f];

% Position analysis with Newton-Raphson method
[q, J, PHI] = position(q, x, l);
[q_f, J_f, PHI_f] = position(q_f, x, l);    % fake model

% Velocity problem
q_dot = velocity(q, phi_dot, x, l);
q_dot_f = velocity(q_f, phi_dot_f, x, l);    % fake model

% Acceleration problem
[q_2dot, B] = acceleration(q, q_dot, phi_2dot, x, l);
[q_2dot_f, B_f] = acceleration(q_f, q_dot_f, phi_2dot_f, x, l);
      %fake model

%% Mass and force matrices

% Gravity constant
g = 9.806;
g_f = 9;%8;   % fake model (gravity is the wrong parameter)

% Calculate mass and force matrices
[M,Q] = mass_forces(m, l, g);
[M_f,Q_f] = mass_forces(m_f, l, g_f);   % fake model

% Adding torque on the crank (DOF) ONLY IN THE REL SYSTEM, NOT IN
     THE MB
% MODEL!!!
Q(5,1) = -10;  % [Nm]

%% Dynamic simulation

% Simulation time parameters
sim_time = 10;
Ts = 5e-3; % sampling
frame = 5e-2;   % plot frame
t = 0:Ts:sim_time-Ts;
```

```
% Kalman filter initializing
[S, R, P, F, H, inn] = observer_init(sim_time, Ts);

% Constraint vectors initializing
PHI = norm(PHI);
PHI_DOT = norm(zeros(4,1));
PHI_2DOT = norm(PHI_DOT);

%Initial values of energy
[ep,ek] = energy(q, q_dot, m, g, x, M);

% Initial values of position and velocity
pos = q;
pos_KF = q_f;
pos_f = q_f;
vel = q_dot;
vel_KF = q_dot_f;
vel_f = q_dot_f;

% Put into KF the same conditions of fake model
q_KF = q_f;
q_dot_KF = q_dot_f;
q_2dot_KF = q_2dot_f;

% Initialize full model matrix eigenvalues vector
eigen = [];
eigen_abs = [];


for k=0.005:Ts:sim_time-Ts

% Adding torque pulse on DOF
if k>4 && k<5
Q(5,1) = 100;
else
Q(5,1) = -10;
end

% Model dynamics calculation
[q, q_dot, q_2dot, constr, constr_p, constr_pp] = augm_lagr3(q,
    q_dot, q_2dot, Ts, M, Q, x, l);
[q_f, q_dot_f, q_2dot_f, ~, ~, ~] = augm_lagr3(q_f, q_dot_f,
    q_2dot_f, Ts, M_f, Q_f, x, l);    % fake model

% Constraint vector updating (to check correctness)
PHI = [PHI norm(constr)];                 % pos
PHI_DOT = [PHI_DOT norm(constr_p)];       % vel
PHI_2DOT = [PHI_2DOT norm(constr_pp)];    % acc

% Run the observer
```

```
[q_KF, q_dot_KF, q_2dot_KF, inn_upd, P, L] = observer(q, q_KF,
    q_dot_KF, q_2dot_KF, Ts, F, P, S, R, H, x, l, Q_f, M_f);

% State space state transition matrix calculation
A = statespace(M, q, q_dot, q_2dot, x, l, Ts);
A_f = statespace(M_f, q_KF, q_dot_KF, q_2dot_KF, x, l, Ts);

% Check filter poles stability
[eigen_new, eigen_abs_new] = eigenvalues(A, A_f, L, H, H, F);

% Eigenvalues vector updating
eigen = [eigen eigen_new];
eigen_abs = [eigen_abs eigen_abs_new];

%Innovation vector updating
inn(end+1)=inn_upd;

% Evaluating energies and updating vectors
[Ep,Ek] = energy(q, q_dot, m, g, x, M);
ep = [ep Ep];        % potential
ek = [ek Ek];        % kinetic

%    % Graphic part
%    if mod(k,frame) == 0 % plots every 0.1 s
%    graphics(q, q_f, q_KF, x, l, 'b', 'r', 'g')
%    end

% Updating vectors
pos = [pos q];
pos_KF = [pos_KF q_KF];
pos_f = [pos_f q_f];
vel = [vel q_dot];
vel_KF = [vel_KF q_dot_KF];
vel_f = [vel_f q_dot_f];
end

%Checking if innovation is white noise
fs = 1/Ts;  % sampling freq.
periodogramma(inn(1,:), fs, 5, 20)%, 'position innovation')

% Plotting energies, if there's conservation the model is well
    defined
figure()
plot(ep)
hold on
plot(ek,'r')
plot(ep+ek,'g')
legend('potential','kinetic','tot mech.')
title('Mechanical energies of the system')

% Plotting constraint vector norm, if the dynamic formulation is
    good its
```

```
% value should be ~= 0
figure()
plot(PHI)
hold on
plot(PHI_DOT,'r')
plot(PHI_2DOT,'g')
title('Constraint vectors')
legend('position','velocity','acceleration')

%%

% Defining estimation errors
e_x1 = pos(1,:) - pos_KF(1,:);  % pos
e_y1 = pos(2,:) - pos_KF(2,:);
e_x2 = pos(3,:) - pos_KF(3,:);
e_y2 = pos(4,:) - pos_KF(4,:);
e_x1_f = pos(1,:) - pos_f(1,:);  % pos
e_y1_f = pos(2,:) - pos_f(2,:);
e_x2_f = pos(3,:) - pos_f(3,:);
e_y2_f = pos(4,:) - pos_f(4,:);
e_x1_dot = vel(1,:) - vel_KF(1,:);  % vel
e_y1_dot = vel(2,:) - vel_KF(2,:);
e_x2_dot = vel(3,:) - vel_KF(3,:);
e_y2_dot = vel(4,:) - vel_KF(4,:);
e_x1_dot_f = vel(1,:) - vel_f(1,:);  % vel
e_y1_dot_f = vel(2,:) - vel_f(2,:);
e_x2_dot_f = vel(3,:) - vel_f(3,:);
e_y2_dot_f = vel(4,:) - vel_f(4,:);

%%

% Plotting estimation performance: x1
figure()
subplot(2,1,1)
g = plot(t, pos(1,:),'k')
hold on
h = plot(t, pos_KF(1,:),'r--')
l = plot(t, pos_f(1,:),'b')
title('Simulation results')
legend('real mechanism','observer','MB model','Location','
    NorthEast','Orientation','Horizontal')
ylabel('$x_1$ coordinate [$m$]','interpreter','latex')
xlabel('Time [$s$]','interpreter','latex')
set(0,'defaultaxesfontsize',18)
set(g(1),'linewidth',3)
set(h(1),'linewidth',3)
set(l(1),'linewidth',3)
set(gca,'box','off')
legend('boxoff')
subplot(2,1,2)
g = plot(t,e_x1(1,:),'r')
hold on
```

```
h = plot(t,e_x1_f(1,:),'b')
legend('real model','observer','Location','SouthWest','
    Orientation','Horizontal')
ylabel('$x_1$ coordinate error [$m$]','interpreter','latex')
xlabel('Time [$s$]','interpreter','latex')
set(0,'defaultaxesfontsize',18)
set(g(1),'linewidth',3)
set(h(1),'linewidth',3)
set(gca,'box','off')
legend('boxoff')

x1max = max(abs(e_x1(20:end)))
x1max2 = max(abs(e_x1_f(20:end)))
x1rms = rms(e_x1(20:end))
x1rms2 = rms(e_x1_f(20:end))
%%

% Plotting estimation performance: y1
figure()
subplot(2,1,1)
g = plot(t, pos(2,:),'k')
hold on
h = plot(t, pos_KF(2,:),'r--')
l = plot(t, pos_f(2,:),'b')
title('Simulation results')
legend('real mechanism','observer','MB model','Location','
    NorthEast','Orientation','Horizontal')
ylabel('$y_1$ coordinate [$m$]','interpreter','latex')
xlabel('Time [$s$]','interpreter','latex')
set(0,'defaultaxesfontsize',18)
set(g(1),'linewidth',3)
set(h(1),'linewidth',3)
set(l(1),'linewidth',3)
set(gca,'box','off')
legend('boxoff')
subplot(2,1,2)
g = plot(t,e_y1,'r')
hold on
h = plot(t,e_y1_f,'b')
legend('real model','observer','Location','SouthWest','
    Orientation','Horizontal')
ylabel('$y_1$ coordinate error [$m$]','interpreter','latex')
xlabel('Time [$s$]','interpreter','latex')
set(0,'defaultaxesfontsize',18)
set(g(1),'linewidth',3)
set(h(1),'linewidth',3)
set(gca,'box','off')
legend('boxoff')

y1max = max(abs(e_y1(20:end)))
y1max2 = max(abs(e_y1_f(20:end)))
y1rms = rms(e_y1(20:end))
```

```
y1rms2 = rms(e_y1_f(20:end))
```

A.1.2 *Graphic function*

```
function [ ] = graphics(q, q_f, q_KF, x, l, color1, color2,
    color3)

% Plots a four bar linkage mechanism

% Natural coordinates
x1 = q(1,1);
y1 = q(2,1);
x2 = q(3,1);
y2 = q(4,1);

% Natural coordinates (fake model)
x1_f = q_f(1,1);
y1_f = q_f(2,1);
x2_f = q_f(3,1);
y2_f = q_f(4,1);

% Natural coordinates (observer)
x1_KF = q_KF(1,1);
y1_KF = q_KF(2,1);
x2_KF = q_KF(3,1);
y2_KF = q_KF(4,1);

% Fixed points
xa = x(1,1);
ya = x(2,1);
xb = x(3,1);
yb = x(4,1);

% Bar length
L1 = l(1,1);

% Plot
figure(10)
clf;
hold on
plot ([xa, x1, x2, xb],[ya, y1, y2, yb], color1 ,  '
    markeredgecolor','k', 'markerfacecolor','g', 'linewidth', 3)
plot ([xa, x1_f, x2_f, xb],[ya, y1_f, y2_f, yb], color2 ,  '
    markeredgecolor','k', 'markerfacecolor','g', 'linewidth', 3)
plot ([xa, x1_KF, x2_KF, xb],[ya, y1_KF, y2_KF, yb], color3 ,  '
    markeredgecolor','k', 'markerfacecolor','g', 'linewidth', 3)
axis equal
xlim ([xa-1.2*L1, xb+0.1*xb])
ylim ([-0.2*xb+xa-1.2*L1, -0.2*xb+xb+0.1*xb])
title('Four-bar linkage mechanism')
xlabel('x coordinate [$m$]', 'interpreter','latex')
ylabel('y coordinate [$m$]', 'interpreter','latex')
```

```
legend('real model','fake model','observer')
legend('boxoff');
set(0,'defaultaxesfontsize',18)
set(gca,'box','off')
drawnow;

end
```

A.1.3 *Energy calculation function*

```
function [Ep,Ek] = energy(q, q_dot, m, g, x, M)

% Evaluates potential and kinetic energy of the whole system

% Fixed point coordinates
ya = x(2,1);
yb = x(4,1);

% q coordinates
y1 = q(2,1);
y2 = q(4,1);

% Bar masses
m1 = m(1,1);
m2 = m(2,1);
m3 = m(3,1);

% Potential energy
Ep = 0.5*g*(m1*(y1+ ya)+m2*(y1+y2)+m3*(y2+yb));

% Kinetic energy
Ek = 0.5*q_dot'*M*q_dot;

end
```

A.2 KINEMATIC FUNCTIONS

A.2.1 *Mechanism initialization function*

```
function [l, m, m_f, x] = kinematic_init()

% Provide phisical information about the mechanism

% Fixed points coordinates
xa = 0;
ya = 0;
xb = 10;
yb = 0;

% Bar lengths
L1 = 2;
L2 = 8;
```

```
L3 = 5;

% Bar masses
m1 = 2;
m2 = 8;
m3 = 5;

% Fake bar mass
m2_f = 5;

% Build full vectors
x = [xa; ya; xb; yb];
m = [m1; m2; m3];
m_f = [m1; m2_f; m3];
l = [L1; L2; L3];

end
```

A.2.2 *Position kinematic problem function*

```
function [q, J, PHI] = position(q, x, l)

% Solve position problem with Newton-Raphson method

% Defining NR method constant
Nmax = 1000; % max iteration n.
tol = 1e-15; % tolerance
e = 1;  % first iteration error
k = 1;  % iteration n.

% Initial position problem (NR)
while e>tol && k<Nmax

%Calculate jacobian and constraint equations matrices
[J, PHI] = jac_constr(q, x, l);

diff = J(:,1:4)\PHI;   % increment
q(1:4,1) = q(1:4,1) - diff;   % new free coordinates value
e = norm(PHI);  % error
k = k+1;    % iteration counter updating
end

end
```

A.2.3 *Velocity kinematic problem function*

```
function [q_dot] = velocity(q, phi_dot, x, l)

% Solve velocity problem

% Check phi: the known terms column varies depending on its value
[~, truccoA, ~] = trick(q, x, l);
```

```
% Jacobian calculation
[J, ~] = jac_constr(q, x, l);

% Velocity analysis: known factors matrix
A = -phi_dot*[0; 0; 0; truccoA(end)];

% Calculate velocities vector
q_dot = J(:,1:4)\A;

% Full velocities vector
q_dot = [q_dot; phi_dot];

end
```

A.2.4 *Acceleration kinematic problem function*

```
function [q_2dot, B] = acceleration(q, q_dot, phi_2dot, x, l)

% Performs acceleration analysis

% Check value of phi
[~, truccoA, ~] = trick(q, x, l);

% Jacobian calculation
[J, ~] = jac_constr(q, x, l);

% Known factors matrix
A = [0; 0; 0; truccoA(end)];

% Calculate jacobian time derivative matrix
B = jac_derivative(q, q_dot, x, l);

% Calculate acceleration vector
q_2dot = J(:,1:4)\(-B*q_dot -phi_2dot*A);

% Build full acceleration vector
q_2dot = [q_2dot; phi_2dot];

end
```

A.2.5 *Trick function*

```
function[trucco, truccoA, truccoB] = trick(q, x, l)

% Checks crank angle (phi) value: if 45<phi<135 deg
% or 225<phi<315 deg use cosine equation , if not use
% sine one

% Fixed point coordinates
xa = x(1,1);
ya = x(2,1);
```

```
% Bar length
L1 = l(1,1);

% DOF value
phi = q(5,1);

% Position of first joint
x1 = q(1,1);
y1 = q(2,1);

% Check phi value
if abs(sin(phi)) > 0.7071
trucco = (x1-xa)-L1*cos(phi);                   % Fourth
    constraint equation (must be equal to zero)
truccoA = [1 0 0 0 L1*sin(phi)];                % Jacobian last
    row (= J(4,:))
truccoB = L1*cos(phi);                          % Jacobian time
    derivative term (= B(4,5))
else
trucco = (y1-ya)-L1*sin(phi);
truccoA = [0 1 0 0 -L1*cos(phi)];
truccoB = L1*sin(phi);
end
end
```

A.2.6 *Jacobian function*

```
function[J, PHI] = jac_constr(q, x, l)

% Compute Jacobian and constraint equations matrices

%  Natural coordinates
x1 = q(1,1);
y1 = q(2,1);
x2 = q(3,1);
y2 = q(4,1);

% Fixed point coordinates
xa = x(1,1);
ya = x(2,1);
xb = x(3,1);
yb = x(4,1);

% Bar lengths
L1 = l(1,1);
L2 = l(2,1);
L3 = l(3,1);

% calculate trucco
[trucco, truccoA, ~] = trick(q, x, l);
```

```
% Jacobian matrix
J = [2*(x1-xa) 2*(y1-ya) 0 0 0;...
-2*(x2-x1) -2*(y2-y1) 2*(x2-x1) 2*(y2-y1) 0;...
0 0 -2*(xb-x2) -2*(yb-y2) 0;...
truccoA];

% Constraint equations
a = (x1-xa)^2+(y1-ya)^2-L1^2;
b = (x2-x1)^2+(y2-y1)^2-L2^2;
c = (xb-x2)^2+(yb-y2)^2-L3^2;
d = trucco;

% Constraints vector
PHI = [a; b; c; d];

end
```

A.2.7 *Jacobian time derivative function*

```
function[B] = jac_derivative(q, q_dot, x, l)

% Calculate time derivative of jacobian matrix

% Velocities
x1_dot = q_dot(1,1);
y1_dot = q_dot(2,1);
x2_dot = q_dot(3,1);
y2_dot = q_dot(4,1);
phi_dot = q_dot(5,1);

% calculate trucco (depending on phi value)
[~, ~, truccoB] = trick(q, x, l);

% Calculate jacobian matrix time derivative (named B)
B = [2*x1_dot 2*y1_dot 0 0 0;...
-2*(x2_dot-x1_dot) -2*(y2_dot-y1_dot) 2*(x2_dot-x1_dot) 2*(y2_dot
    -y1_dot) 0;...
0 0 2*x2_dot 2*y2_dot 0;...
0 0 0 0 truccoB*phi_dot];

end
```

A.3 DYNAMIC FUNCTIONS

A.3.1 *Mass and forces calculation function*

```
function [M,Q] = mass_forces(m, l, g)

% Compute mass and generalized force matrices for the mechanism

% Bar mass
m1 = m(1,1);
```

```
m2 = m(2,1);
m3 = m(3,1);

% Bar lengths
L1 = l(1,1);
L2 = l(2,1);
L3 = l(3,1);

% Mass matrices of the three bars
M1 = [m1/3 0 m1/6 0; 0 m1/3 0 m1/6; m1/6 0 m1/3 0; 0 m1/6 0 m1
    /3];
M2 = [m2/3 0 m2/6 0; 0 m2/3 0 m2/6; m2/6 0 m2/3 0; 0 m2/6 0 m2
    /3];
M3 = [m3/3 0 m3/6 0; 0 m3/3 0 m3/6; m3/6 0 m3/3 0; 0 m3/6 0 m3
    /3];

% Build mass matrix of the entire mechanism
M = zeros(5,5);
M(1:2,1:2) = M1 (3:4,3:4);
M(3:4,3:4) = M3 (1:2,1:2);
M(1:4,1:4) = M(1:4,1:4) + M2;

% Force matrices of the three bars (gravity)
Q1 = 1/L1*[L1/2 0; 0 L1/2; L1/2 0; 0 L1/2]*[0;-m1*g];
Q2 = 1/L2*[L2/2 0; 0 L2/2; L2/2 0; 0 L2/2]*[0;-m2*g];
Q3 = 1/L3*[L3/2 0; 0 L3/2; L3/2 0; 0 L3/2]*[0;-m3*g];

% Build force vector for the entire mechanism
Q = zeros(5,1);
Q(1:2,1) = Q1(3:4,1);
Q(3:4,1) = Q3(1:2,1);
Q(1:4,1) = Q(1:4,1) + Q2;

end
```

A.3.2 *Index-3 augmented Lagrange dynamic formulation function*

```
function [q, q_dot, q_2dot, PHI, PHI_DOT, PHI_2DOT] = augm_lagr3(
    q, q_dot, q_dot_dot, Ts, M, Q, x, l)

% Dynamic formulation which uses index-3 augmented Lagrangian
    approach,
% with projections in velocity and accelerations in order to
    fully
% satisfy the constraints.

alpha = 1e9;    % penalty term
tol = 1e-7;     % tolerance
err = 1;        % initial value for the error
Nmax = 1e3;     % iteration max no.
k = 0;          % initialize counter
lambda = 0;     % initialize Lagrange multiplier
```

```
% Precalculating pos, vel, acc: "hat" terms
q_dot_hat = -(2/Ts*q + q_dot);                            % vel
q_2dot_hat = -(4/(Ts^2)*q + 4/Ts*q_dot + q_dot_dot);    % acc

% Precalculating pos, vel, acc: full calculation
q = q + Ts*q_dot + 0.5*Ts^2*q_dot_dot;             % pos
q_dot = (2/Ts)*q + q_dot_hat;                        % vel
q_dot_dot = (4/Ts^2)*q + q_2dot_hat;                 % acc

% Solving the nonlinear problem using Newton-Raphson method
while err > tol && k < Nmax

% Jacobian and constraint equation matrices updating
[J, PHI] = jac_constr(q, x, l);

% Updating Lagrange multiplier value
lambda = lambda + alpha*PHI;

% Calculation of dynamic equilibrium equation
f = Ts^2/4*(M*q_dot_dot + J'*alpha*PHI + J'*lambda - Q);

% Dynamic equilibrium tangent
f_q = M +0.25*Ts^2*(J'*alpha*J);

% Delta calculation
deltaq = f_q\-f;

% Updating calculation of position, velocity, acceleration
q = q + deltaq;                            % pos
q_dot = 2/Ts*q + q_dot_hat;                    % vel
q_dot_dot = 4/(Ts^2)*q + q_2dot_hat;        % acc

% Error calculation
err = norm(deltaq);

% Counter updating
k = k+1;
end

% Final matrices updating
[J, PHI] = jac_constr(q, x, l);

% Velocity projection
q_dot = f_q\(M*q_dot);

% Update velocity constraint
PHI_DOT = J*q_dot;

% Update Jacobian time derivative
B = jac_derivative(q, q_dot, x, l);
```

```
% Acceleration projection
q_2dot = f_q\(M*q_dot_dot - Ts^2/4*J'*alpha*B*q_dot);

% Acceleration constraints calculation
PHI_2DOT = J*q_2dot + B*q_dot;

end
```

A.3.3 *Integration function*

```
function[q_new, q_dot_new, q_2dot_new, PHI, PHI_DOT, PHI_2DOT] =
    integrate(q, q_dot, q_2dot, Ts, x, l, Q ,M)

% Perform calculation of new positions and velocities through
    acceleration
% integration, using trapezoidal rule

% Error tolerance into the numerical integration
tol = 1e-5;

% First estimations at step k+1 (forward Euler)
v = q_dot + q_2dot*Ts;  % velocity prediction
p = q + q_dot*Ts +Ts^2/4*q_2dot;   % position prediction

% Initial error values
err_v = 1;
err_p = 1;

% Initial value for Lagrange multiplier
lambda = 0;

% Trapezoidal integration loop
while err_p > tol || err_v > tol

% Matrices updating (with position and velocities at step k)
[J, PHI] = jac_constr(p, x, l);
B = jac_derivative(p, v, x, l);

% Calculate accelerations step k+1 with penalty formulation
[q_dot_dot, lambda, alpha, PHI_DOT] = augm_lagr3(J, p, v, lambda,
     B, M, Q, PHI);

% Velocity calculation with trapezoidal rule (k+1)
v_new = q_dot + Ts/2*(q_2dot + q_dot_dot);


% Position calculation with trapezoidal rule (k+1)
p_new = q + Ts*q_dot + Ts^2/4*(q_2dot + q_dot_dot);

% Check the errors through the norm of the vectors
err_v = norm(v - v_new);      % velocity
err_p = norm(p - p_new);      % position
```

```
%Update vectors
v = v_new;
p = p_new;

end

% Final matrices updating
[J, PHI] = jac_constr(p, x, l);
B = jac_derivative(p, v, x, l);

% Output vectors
q_new = p;
q_dot_new = v;
q_2dot_new = q_dot_dot;
PHI_DOT = J*q_dot_new;
PHI_2DOT = J*q_2dot_new + B*q_dot_new;

end
```

A.4 OBSERVER FUNCTIONS

A.4.1 *Observer initialization function*

```
function [S, R, P, F, H, inn] = observer_init(sim_time, Ts)

% Initialization of the observer

% Filter initialization: model noise
n_pos = 1; % position [deg]
n_vel = 1; % velocity [deg/s]

% Model noise matrix -> TO BE TUNED IN ORDER TO OBTAIN WHITE
    INNOVATION
S11 = degtorad(n_pos)*Ts;       % Multiplication of position
    model noise by Ts, like in F matrix
S22 = degtorad(n_vel)*Ts/5e-3;
S = 1e-1*diag([S11 S22]);

% Measurement noise (encoder uncertainty)
n_meas = 1;

% Measurement noise matrix
R = degtorad(n_meas)^2;

% Setting initial values
P = eye(2);    % covariance
F = [1 Ts; 0 1]; % state transition
H = [1 0];        % output matrix (assuming we have an encoder on
     the crank)
inn = [];        % innovation
```

```
end
```

A.4.2 *Observer function*

```
function[q_KF, q_dot_KF, q_2dot_KF, inn, P_upd, L] = observer(q,
    q_f, q_dot_f, q_2dot_f, Ts, F, P, S, R, H, x, l, Q, M_f)

% Designing an errorEKF state observer, defining the state as the
     vector
% containing the errors in position and velocity:
%
%   s := [e_p e_v]'
%
%   s(k+1) = F*s(k)
%
%   F = [I DT*I]
%       [0    I]
%

% Predicting step
e_pred = zeros(2,1);    % state
P_pred = F*P*F'+S;      % covariance

% One timestep integration: q_KF are positions of the observer at
% k-step, q_f are the same but at k-1 step
[q_KF, q_dot_KF, q_2dot_KF, ~,~,~] = augm_lagr3(q_f, q_dot_f,
    q_2dot_f, Ts, M_f, Q, x, l);

% Updating step: Kalman gain
L = P_pred*H'/(H*P_pred*H'+ R);

% Measurements
noise = sqrt(R)*randn(1);   % Adding noise to the "measurement"
    from the real model, in order to simulate a real sensor
meas = q(5,1) + noise;   % phi from real model, simulated with
    noise
meas_f = q_KF(5,1);  % phi from observer

% Innovation calculation
inn = meas - meas_f;    % innovation at step k

% Updating step
e_upd = e_pred + L*inn; % state
P_upd = (eye(2) - L*H)*P_pred*(eye(2) - L*H)' + L*R*L'; %
    covariance

% Computing errors through velocity problem
e_pos = velocity(q_KF, e_upd(1,1), x, l);    % position  QUI C'E'
     IL TRUCCO (*)
e_vel = velocity(q_KF, e_upd(2,1), x, l);    % velocity

% Model correction
```

```
q_KF = q_KF + e_pos;  % position
q_dot_KF = q_dot_KF + e_vel;  % velocity

end

%(*) Gli errori di posizione vengono di fatto calcolati
    utilizzando la
% formulazione per il problema di velocità. q_KF son le posizioni
     stimate
% dall'osservatore al passo corrente, e_upd è l'errore di
    posizione
% (utilizzato al posto della coordinata libera di velocità). x e
    l sono i
% parametri fisici del quadrilatero
```

A.4.3 *Mechanism state-transition matrix calculation function*

```
function [A] = statespace(M, q, q_dot, q_2dot, x, l, Ts)

% Computes the state space formulation - state transition matrix
    - of
% the mechanism through R matrix method (see "Real-time state
    observers
% based on multibody models and the extended Kalman filter",
    Cuadrado
% et al., Journal of Mechanical Science and Technology 23 (2009)
% 894~900).

% Defining variables
% Natural coordinates
x1 = q(1,1);
y1 = q(2,1);
x2 = q(3,1);
y2 = q(4,1);
phi = q(5,1);
phi_2dot = q_2dot(5,1);

% Fixed points
xa = x(1,1);
ya = x(2,1);
xb = x(3,1);
yb = x(4,1);

% R matrix calculation: solving velocity problem with DOF
    velocity = 1
R = velocity(q, 1, x, l);


% Calculate different Rq depending on phi value (another "trucco
    ").  Rq
% is an hypermatrix containing R matrix partial derivatives wrt
    natural
```

```
% coordinates.  Rq has been calculated using Matlab script "R.m",
     in symbolic mode.
if abs(sin(phi)) > 0.7071
Rq = [

0,

    0,

    0,

    0,

    - sin(phi) - phi*cos(phi);...
(phi*sin(phi))/(y1 - ya),

    -(phi*x1*sin(phi) - phi*xa*sin(phi))/(y1 - ya)^2,

    0,

    0,

    (x1*sin(phi) - xa*sin(phi) + phi*x1*cos(phi) - phi*xa*cos(phi
    ))/(y1 - ya);...
(phi*sin(phi)*(y2 - ya)*(y2 - yb))/(x2*y1^2 - xb*y1^2 - x1*y1*y2
    + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*
    ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya) + (phi*sin(phi)*(y2 -
     yb)*(y1*y2 - y2*ya - y1*yb + ya*yb)*(x1*y2 - x2*y1 - x1*ya +
     xa*y1 + x2*ya - xa*y2))/(x2*y1^2 - xb*y1^2 - x1*y1*y2 + x1*
    y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*ya*yb
    + xb*y1*ya + x2*ya*yb - xb*y2*ya)^2, (phi*sin(phi)*(y2 - yb)
    *(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2)*(x1*y2 - 2*
    x2*y1 + x2*ya - x1*yb + 2*xb*y1 + x2*yb - xb*y2 - xb*ya))/(x2
    *y1^2 - xb*y1^2 - x1*y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb -
     x2*y1*yb + xb*y1*y2 - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2
    *ya)^2 - (phi*sin(phi)*(x2 - xa)*(y2 - yb))/(x2*y1^2 - xb*y1
    ^2 - x1*y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb +
    xb*y1*y2 - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya),

    (phi*sin(phi)*(y2 - yb)*(y1*ya + y1*yb - ya*yb - y1^2)*(x1*y2
     - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/(x2*y1^2 - xb*y1^2
     - x1*y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*
    y1*y2 - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya)^2 - (phi*
    sin(phi)*(y1 - ya)*(y2 - yb))/(x2*y1^2 - xb*y1^2 - x1*y1*y2 +
     x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*ya
    *yb + xb*y1*ya + x2*ya*yb - xb*y2*ya), (phi*sin(phi)*(x1*y2 -
     x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/(x2*y1^2 - xb*y1^2 -
     x1*y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1
    *y2 - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya) + (phi*sin(
    phi)*(x1 - xa)*(y2 - yb))/(x2*y1^2 - xb*y1^2 - x1*y1*y2 + x1*
    y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*ya*yb
    + xb*y1*ya + x2*ya*yb - xb*y2*ya) + (phi*sin(phi)*(y2 - yb)*(
    x1*y1 - x1*ya - xb*y1 + xb*ya)*(x1*y2 - x2*y1 - x1*ya + xa*y1
     + x2*ya - xa*y2))/(x2*y1^2 - xb*y1^2 - x1*y1*y2 + x1*y2*ya -
     x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*ya*yb + xb*y1
    *ya + x2*ya*yb - xb*y2*ya)^2,   (sin(phi)*(y2 - yb)*(x1*y2 -
    x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/(x2*y1^2 - xb*y1^2 -
    x1*y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*
    y2 - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya) + (phi*cos(
    phi)*(y2 - yb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2
    ))/(x2*y1^2 - xb*y1^2 - x1*y1*y2 + x1*y2*ya - x2*y1*ya + x1*
    y1*yb - x2*y1*yb + xb*y1*y2 - x1*ya*yb + xb*y1*ya + x2*ya*yb
    - xb*y2*ya);...
- (phi*sin(phi)*(x2 - xb)*(y2 - ya))/(x2*y1^2 - xb*y1^2 - x1*y1*
    y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 -
    x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya) - (phi*sin(phi)*(
    x2 - xb)*(y1*y2 - y2*ya - y1*yb + ya*yb)*(x1*y2 - x2*y1 - x1*
    ya + xa*y1 + x2*ya - xa*y2))/(x2*y1^2 - xb*y1^2 - x1*y1*y2 +
    x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*ya*
    yb + xb*y1*ya + x2*ya*yb - xb*y2*ya)^2, (phi*sin(phi)*(x2 -
    xa)*(x2 - xb))/(x2*y1^2 - xb*y1^2 - x1*y1*y2 + x1*y2*ya - x2*
    y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*ya*yb + xb*y1*ya
    + x2*ya*yb - xb*y2*ya) - (phi*sin(phi)*(x2 - xb)*(x1*y2 - x2*
    y1 - x1*ya + xa*y1 + x2*ya - xa*y2)*(x1*y2 - 2*x2*y1 + x2*ya
    - x1*yb + 2*xb*y1 + x2*yb - xb*y2 - xb*ya))/(x2*y1^2 - xb*y1
    ^2 - x1*y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb +
    xb*y1*y2 - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya)^2, (phi
    *sin(phi)*(x2 - xb)*(y1 - ya))/(x2*y1^2 - xb*y1^2 - x1*y1*y2
    + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*
    ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya) - (phi*sin(phi)*(x1*
    y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/(x2*y1^2 - xb*y1
    ^2 - x1*y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb +
    xb*y1*y2 - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya) - (phi*
    sin(phi)*(x2 - xb)*(y1*ya + y1*yb - ya*yb - y1^2)*(x1*y2 - x2
    *y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/(x2*y1^2 - xb*y1^2 - x1
    *y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2
     - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya)^2,

    - (phi*sin(phi)*(x1 - xa)*(x2 - xb))/(x2*y1^2 - xb*y1^2 - x1*
    y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2
    - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya) - (phi*sin(phi)
    *(x2 - xb)*(x1*y1 - x1*ya - xb*y1 + xb*ya)*(x1*y2 - x2*y1 -
    x1*ya + xa*y1 + x2*ya - xa*y2))/(x2*y1^2 - xb*y1^2 - x1*y1*y2
     + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*
    ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya)^2, - (sin(phi)*(x2 -
    xb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/(x2*y1^2
     - xb*y1^2 - x1*y1*y2 + x1*y2*ya - x2*y1*ya + x1*y1*yb - x2*
    y1*yb + xb*y1*y2 - x1*ya*yb + xb*y1*ya + x2*ya*yb - xb*y2*ya)
     - (phi*cos(phi)*(x2 - xb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 +
    x2*ya - xa*y2))/(x2*y1^2 - xb*y1^2 - x1*y1*y2 + x1*y2*ya - x2
    *y1*ya + x1*y1*yb - x2*y1*yb + xb*y1*y2 - x1*ya*yb + xb*y1*ya
     + x2*ya*yb - xb*y2*ya);...
0,
    0,

    0,

    0,

    0];



else
Rq = [

(phi*cos(phi)*(y1 - ya))/(x1 - xa)^2,

    -(phi*cos(phi))/(x1 - xa),

    0,

    0,

    (phi*sin(phi)*(y1 - ya))/(x1 - xa) - (cos(phi)*(y1 - ya))/(x1
     - xa);...
0,

    0,

    0,

    0,

    cos(phi) - phi*sin(phi);...
(phi*cos(phi)*(y2 - yb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya -
    xa*y2))/((x1 - xa)^2*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb -
     xb*y2)) + (phi*cos(phi)*(y2 - yb)^2*(x1*y2 - x2*y1 - x1*ya +
     xa*y1 + x2*ya - xa*y2))/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb +
    xb*y1 + x2*yb - xb*y2)^2) - (phi*cos(phi)*(y2 - ya)*(y2 - yb)
    )/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2))
    , (phi*cos(phi)*(x2 - xa)*(y2 - yb))/((x1 - xa)*(x1*y2 - x2*
    y1 - x1*yb + xb*y1 + x2*yb - xb*y2)) - (phi*cos(phi)*(x2 - xb
    )*(y2 - yb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))
    /((x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2)
    ^2),

    (phi*cos(phi)*(y1 - ya)*(y2 - yb))/((x1 - xa)*(x1*y2 - x2*y1
    - x1*yb + xb*y1 + x2*yb - xb*y2)) - (phi*cos(phi)*(y1 - yb)*(
    y2 - yb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/((
    x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2)^2),
    (phi*cos(phi)*(x1 - xb)*(y2 - yb)*(x1*y2 - x2*y1 - x1*ya + xa
    *y1 + x2*ya - xa*y2))/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*
    y1 + x2*yb - xb*y2)^2) - (phi*cos(phi)*(x1*y2 - x2*y1 - x1*ya
     + xa*y1 + x2*ya - xa*y2))/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb
    + xb*y1 + x2*yb - xb*y2)) - (phi*cos(phi)*(y2 - yb))/(x1*y2 -
     x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2), (phi*sin(phi)*(y2 -
    yb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/((x1 -
    xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2)) - (cos(
    phi)*(y2 - yb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2
    ))/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2)
    );...
(phi*cos(phi)*(x2 - xb)*(y2 - ya))/((x1 - xa)*(x1*y2 - x2*y1 - x1
    *yb + xb*y1 + x2*yb - xb*y2)) - (phi*cos(phi)*(x2 - xb)*(x1*
    y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/((x1 - xa)^2*(x1
    *y2 - x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2)) - (phi*cos(phi)
    *(x2 - xb)*(y2 - yb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya -
     xa*y2))/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb -
    xb*y2)^2),         (phi*cos(phi)*(x2 - xb)^2*(x1*y2 - x2*y1 -
     x1*ya + xa*y1 + x2*ya - xa*y2))/((x1 - xa)*(x1*y2 - x2*y1 -
    x1*yb + xb*y1 + x2*yb - xb*y2)^2) - (phi*cos(phi)*(x2 - xa)*(
    x2 - xb))/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb -
     xb*y2)), (phi*cos(phi)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*
    ya - xa*y2))/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*
    yb - xb*y2)) - (phi*cos(phi)*(x2 - xb)*(y1 - ya))/((x1 - xa)
    *(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2)) + (phi*cos(
    phi)*(x2 - xb)*(y1 - yb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*
    ya - xa*y2))/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*
    yb - xb*y2)^2),

    (phi*cos(phi)*(x2 - xb))/(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*
    yb - xb*y2) - (phi*cos(phi)*(x1 - xb)*(x2 - xb)*(x1*y2 - x2*
    y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/((x1 - xa)*(x1*y2 - x2*
    y1 - x1*yb + xb*y1 + x2*yb - xb*y2)^2), (cos(phi)*(x2 - xb)*(
    x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2))/((x1 - xa)*(
    x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2)) - (phi*sin(
    phi)*(x2 - xb)*(x1*y2 - x2*y1 - x1*ya + xa*y1 + x2*ya - xa*y2
    ))/((x1 - xa)*(x1*y2 - x2*y1 - x1*yb + xb*y1 + x2*yb - xb*y2)
    );...
0,

    0,

    0,

    0,

    0];


end

% R matrix time derivative calculation
R_dot = Rq*q_dot;

%Mass matrix projection
Mr = R'*M*R;
```

```
% State transition matrix
A21 = -Mr\(R'*2*M*Rq*R*phi_2dot);
A22 = -Mr\(R'*M*R_dot);
A = [0 1; A21 A22];

% A matrix discretization
A = expm(A*Ts);
end
```

A.4.4 *Observer poles stability analysis function*

```
function [eigen, eigen_abs] = eigenvalues(A, A_f, L, H, H_f, F)

% Perform a pole analysis of the full state transition matrix of
    the
% system, including MB model and oserver.  The entire state has
    been
% defined as:
%
%   s := [x x_MB e]', where:
%
%   x = [phi phi_dot];
%   x_MB = [phi_f phi_dot_f]'
%   e = [e_phi e_phi_dot]'

% Define model size
s = 2;

% Define eigenvalues vector
eigen = zeros(3*s,1);
eigen_abs = zeros(3*s,1);

% Full model state transition matrix

% Total gain
K = F*L;
KSTAR = [K(1,1) 0; 0 K(2,1)];

% Build full state-transition matrix
G = [A zeros(s) zeros(s);...
zeros(s) A_f 0;...      %F*KSTAR                    %F*L(:,1)*H
F*L(:,1)*H -F*L(:,1)*H_f zeros(s)];

% Eigenvalues calculation
eigen(:,1) = eig(G);

for k = 1:3*s

% Compute magnitude of every eigenvalue
eigen_abs(k,1) = abs(eigen(k,1));
```

```
% Check stability
if eigen_abs(k,1) >= 1

disp('Unstable')
end
end

end
```

A.4.5 *Periodogram*

```
function periodogramma(x, Fs, tol, j) %ogg)

%   Performs cumulative periodogram calculation of x
%
%   x : data
%   Fs: sampling freq.
%   tol: tolerance (%)
%   j: figure number (just for sake of clarity)
%   ogg: peridogram of...? (string)

% Data number (even number)
N = max(size(x));
if ceil(N/2)*2 ~= N
N = N-1;
x = x(1:N);
end

% sequnce DFT
X = fft(x,N);

% periodogram
Sx = ((abs(X)).^2)/N;

% cumulative periodogram
Px = zeros(N/2,1);
Px(1)=Sx(1);

for k = 2:N/2

Px(k) = Px(k-1) + 2*Sx(k);
end

Px = Px/N;

% variance estimation
mx = sum(x)/N;
var2x = sum((x-mx).^2)/N;

% ideal cumulative periodogram
I = zeros(N/2,1);
I(1)=1;
```

```
for k = 2:N/2

I(k) = 2*k;
end

Ix = (I*var2x)./N;
Itolp = ( (I*var2x) )./N + var2x*(tol/100);
Itoln = ( (I*var2x) )./N - var2x*(tol/100);

% periodogram plot
figure(j);
subplot(2,1,1)
hold on
f = (1:N/2)*(Fs/(N/2));
g = plot(f,Ix,'b');
l = plot(f,Px,'r--');
h = plot(f,Itolp,'k:');
k = plot(f,Itoln,'k:');
hold off

title(['Cumulative Periodogram']);%, ogg]);
xlabel('Frequency [$Hz$]','interpreter','latex');
legend('ideal periodogram', 'real periodogram', 'tolerance' ,2, '
    Location','SouthEast');
set(0,'defaultaxesfontsize',18)
set(g(1),'linewidth',3)
set(l(1),'linewidth',3)
set(h(1),'linewidth',3)
set(k(1),'linewidth',3)
set(gca,'box','off')
legend('boxoff')

subplot(2,1,2)
hold on
l = plot(f,Sx(1:N/2),'r--')
g = plot(f,ones(size(f))*var2x);
hold off
set(0,'defaultaxesfontsize',18)
set(g(1),'linewidth',3)
set(l(1),'linewidth',3)
set(gca,'box','off')
xlabel('Frequency [$Hz$]','interpreter','latex');


return
```

BIBLIOGRAPHY

[1] M. Ahmadi, A. Khayatian, and P. Karimaghaee. Orientation Estimation by Error-State Extended Kalman Filter in Quaternion Vector Space. In *Proceedings of SICE Annual Conference*, Kagawa, Japan, September 2007.

[2] F. Aparicio, J. Paez, F. Moreno, F. Jiménez, and A. López. Discussion of a new adaptive control system incorporating the geometric characteristic of the roadway. *International Journal of Vehicle Autonomus Systems*, 3(1):47–64, January 2005.

[3] M. Athans. *The Control Handbook*, chapter Kalman Filtering, pp. 589–594. CRC Press LLC, 1996.

[4] F. Battaglia. *Metodi di Previsione Statistica*. Springer-Verlag, Milano, 2007.

[5] E. Bayo and R. Ledesma. Augmented Lagrangian and Mass-Orthogonal Projection Methods for Constrained Multibody Dynamics. *Nonlinear Dynamics*, 9:113–130, February 1996.

[6] F. Cheli and E. Pennestrì. *Cinematica e Dinamica dei Sistemi Multibody*. Casa Editrice Ambrosiana, Milano, 2006.

[7] J. Cuadrado, R. Gutiérrez, M. A. Naya, and P. Morer. A comparison in terms of accuracy and efficiency between a MBS dynamic formulation with stress analysis and a non-linear FEA code. *International Journal for Numerical Methods in Engineering*, 51:1033–1052, April 2001.

[8] J. Cuadrado, D. Dopico, A. Barreiro, and E. Delgado. Real-time state observers based on multibody models and the extended Kalman filter. *Journal of Mechanical Science and Technology*, 23: 894–900, April 2009.

[9] D. Dopico, J. Cuadrado, F. González, and J. Kövecses. Determination of Holonomic and Nonholonomic Constraint Reactions in an Index-3 Augmented Lagrangian Formulation With Velocity and Acceleration Projections. *Journal of Computational and Nonlinear Dynamics*, 9:041006.1–041006.9, October 2014.

[10] M. Doumiati, A. Charara, A. Victorino, and D. Lechner. *Vehicle Dynamics Estimation using Kalman Filter*. Iste Ltd., London, 2013.

[11] C. M. Farmer. Effects of Electronic Stability Control on Fatal Crash Risk. Technical report, Insurance Institute for Highway Safety, 2010.

[12] E. Fornasini and G. Marchesini. *Appunti di Teoria dei Sistemi*. Ed. Libreria Progetto, Padua, 2003.

[13] J. García de Jalón and E. Bayo. *Kinematic and Dynamic Simulation of Multibody Systems*. Springer-Verlag, New York, 1994.

[14] M. S. Grewal and A. P. Andrews. *Kalman Filtering, Theory and Practice using Matlab*. John Wiley & Sons, Inc, Hoboken, NJ, 2008.

[15] R. Hermann and A. J. Krener. Nonlinear Controllability and Observability. *IEEE Transactions on Automatic Control*, 22(5):728–740, October 1977.

[16] S. J. Julier and J. K. Uhlmann. Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, 92(3):401–422, March 2004.

[17] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, pages 35–45, March 1960.

[18] J. L. Lagrange. *Mécanique Analytique*. M.me V.e Courcier, Imprimeur-Libraire pour les Mathématiques, Paris, 1815.

[19] D. G. Luenberger. Observing the State of a Linear System. *IEEE Transactions on Military Electronics*, 8:74–80, April 1964.

[20] P. Matisko. *Estimation of the stochastic properties of controlled systems*. PhD thesis, Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, May 2013.

[21] D. Mizell. Using Gravity to Estimate Accelerometer Orientation. In *Proceedings of the Seventh IEEE International Symposium on Wearable Computers*, Zurich, Switzerland, September 2013.

[22] World Health Organization. *Global Status Report on Road Safety 2015*. WHO Press, Geneve, 2015. URL http://www.who.int/violence_injury_prevention/road_safety_status/2015/en/.

[23] H. B. Pacejka. *Tyre and Vehicle Dynamics*. Butterworth-Heinemann, Oxford, 2002.

[24] I. Palomba, D. Richiedei, and A. Trevisani. Nonlinear kinematic state estimation in rigid-link multibody systems by spherical simplex sigma point unscented kalman filters. In *Proceedings of the 26th International Conference on Noise and Vibration Engineering*, Leuven, Belgium, September 2014.

[25] I. Palomba, D. Richiedei, and A. Trevisani. Simultaneous estimation of kinematic state and unknown input forces in rigid-link multibody systems. In *Proceedings of ECCOMAS Thematic Conference on Multibody Dynamics*, Barcelona, Catalonia, ES, June 2015.

[26] R. Pastorino. *Experimental Validation of a Multibody Model for a Vehicle Prototype and its Application to Automotive State Observers*. PhD thesis, Departamento de Ingeniería Industrial II, Escuela Politécnica Superior, Universidad de A Coruña, June 2012.

[27] R. Pastorino, D. Richiedei, J. Cuadrado, and A. Trevisani. State estimation using multibody models and non-linear Kalman filters. *International Journal of Non-Linear Mechanics*, 53:83–90, July 2013.

[28] S. I. Roumeliotis, G. S. Sukhatme, and G. A. Bekey. Smoother based 3D Attitude Estimation for Mobile Robot Localization. Technical report, Institute for Robotics and Intelligent Systems, University of Southern California, 1998.

[29] S. I. Roumeliotis, G. S. Sukhatme, and G. A. Bekey. Circumventing Dynamic Modeling: Evaluation of the Error-State Kalman Filter applied to Mobile Robot Localization. In *Proceedings of IEEE International Conference on Robotics & Automation*, Detroit, MI, May 1999.

[30] E. Sanjurjo, J. L. Blanco, J. L. Torres, and M. A. Naya. Testing the efficiency and accuracy of multibody-based state observers. In *Proceedings of ECCOMAS Thematic Conference on Multibody Dynamics*, Barcelona, Catalonia, ES, June 2015.

[31] D. Simon. *Optimal State Estimation*. John Wiley & Sons, Inc, Hoboken, NJ, 2006.

[32] Y. S. Suh. Orientation Estimation Using a Quaternion-Based Indirect Kalman Filter With Adaptive Estimation of External Acceleration. *IEEE Transactions on Instrumentation and Measurement*, 59(12):3296–3305, December 2010.

[33] J. L. Torres, J. L. Blanco, E. Sanjurjo, M. Á. Naya, and A. Giménez. Towards Benchmarking of State Estimators for Multibody Dynamics. In *Proceedings of the 3rd International Conference on Multibody System Dynamics*, BEXCO, Busan, Korea, July 2014.

[34] C. F. Van Loan. Computing Integrals Involving the Matrix Exponential. *IEEE Transactions on Automatic Control*, 23(3):395–404, June 1978.

[35] G. Welch and G. Bishop. An Introduction to the Kalman Filter. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, 2006.