الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

École Nationale Polytechnique
Département d'Automatique

End of studies project thesis

Submitted in fulfillment of the requirements
for the State Engineer Degree in Automation Engineering

Control and Estimation On Lie Groups, Application on Autonomous Underwater vehicles

Realized by:
Mr. DERBAL Mosaab

Supervised by:
Pr. TADJINE Mohamed

Publicly presented and defended on the 30th of June, 2022.

Jury members:

President	Pr El Madjid Berkouk	ENP
Promoter	Pr TADJINE Mohamed	ENP
Examiner	Pr Messaoud CHAKIR	ENP

ENP 2022

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique
Département d'Automatique

End of studies project thesis

Submitted in fulfillment of the requirements
for the State Engineer Degree in Automation Engineering

Control and Estimation On Lie Groups, Application on Autonomous Underwater vehicles

Realized by:
Mr. DERBAL Mosaab

Supervised by:
Pr. TADJINE Mohamed

Publicly presented and defended on the 30th of June, 2022.

Jury members:

President	Pr El Madjid Berkouk	ENP
Promoter	Pr TADJINE Mohamed	ENP
Examiner	Pr Messaoud CHAKIR	ENP

ENP 2022

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

École Nationale Polytechnique

Département Automatique

Mémoire de projet de fin d'études

pour l'obtention du diplôme d'Ingénieur en Automatique

Commande et Estimation sur les groupes de Lie. Application en Robot sous-marin autonome

Réalisé par :
M. DERBAL Mosaab

Supervisé par:
Mr. TADJINE Mohamed

Présenté et soutenue publiquement le 30 Juin 2022.

Membres du jury :

Président	Pr	ENP
Promoteur	MAA	ENP
Examinateur	MAA	ENP

ENP 2022

ملخص

يركز هذا العمل على تقدير ومراقبة الأنظمة الديناميكية التي تتطور في مجموعات لي ، مع تطبيقات في الروبوتات. في الواقع ، تتطور معظم الأنظمة الديناميكية بشكل طبيعي على SO (3) و SE (3). ومع ذلك ، فإن تصميم وحدات التحكم والمراقبين الذين يمكنهم توفير تقدير دقيق ودفع النظام إلى الموضع المطلوب في مجموعات لي هو أمر صعب. يوضح هذا العمل حل كل من هاتين المشكلتين ، من خلال تصميم وحدات تحكم مشتركة مثل متحكم تناسبي تفاضلي ، والوضع الانزلاقي ، و LQR على مجموعات لي، ومن خلال إنشاء ملاحظ الذي يمكنه تقدير الحالات الموجودة في مجموعات لي مثل إصدار مجموعة لي من الإصدار الموسع. مراقب كالمان. أخيرًا ، تتمثل مساهمتي في تنفيذ وحدة تحكم PD تعتمد على مراقب LG-EKF على مركبة مستقلة تحت الماء تتطور على SE (2).

كلمات مفتاحية : تجميع مجموعة لي , مراقب كالمان, مركبة تحت الماء تلقائي.

Résumé

Ce travail se concentre sur l'estimation et la commande de systèmes dynamiques évoluant sur les groupes de Lie, avec des applications en robotique. En effet, la plupart des systèmes dynamiques évoluent naturellement sur SO(3) et SE(3). Cependant, la conception de contrôleurs et d'observateurs capables de fournir une estimation précise et de conduire le système à la position désirée sur les groupes de Lie est assez difficile. Ce travail élabore la solution de ces deux problèmes, en concevant des contrôleurs communs tels que la dérivée proportionnelle, le mode glissant, le LQR sur les groupes de Lie et en construisant des observateurs qui peuvent estimer des états qui existent sur les groupes de Lie tels que la version du filtre de Kalman étendu pour les groupes de Lie. Enfin, ma contribution est l'implémentation d'un contrôleur PD basé sur un observateur LG-EKF sur un véhicule sous-marin autonome évoluant sur SE(2).

Mots clés : Théorie de Lie, Filtre de Kalman étendu, Robot sous-marin autonome

---

Abstract

This work focuses on the estimation and control of dynamical systems evolving on Lie groups, with applications in robotics. Indeed, most dynamical systems evolve naturally on SO(3) and SE(3). However, designing controllers and observers which can provide precise estimation and drive the system to the desired position on Lie groups is quite challenging. This work elaborates on the solution of both of these problems, by designing common controllers such as Proportional Derivative, Sliding Mode, LQR on Lie groups and by constructing Observers which can estimate states that exist on Lie groups such as the Lie Group version of the Extended Kalman Filter. Finally, My contribution is the implementation of An LG-EKF observer-based PD-controller on an autonomous underwater vehicle evolving on SE(2).

Keywords : Lie Group, Extended kalman filter, Autonomous underwater Vehicle.

Dedicace

> “
>
> *To my dear Father and Mother, my family, and my friends.*
>
> ”

- Derbal Mosaab

Remerciments

First of all, I thank God the Almighty for giving me the courage, the will and the patience to carry out this work.

I thank my parents, who supported me throughout my studies' journey.

I would like to express my gratitude to my supervisor, Pr. Mohamed TADJINE, for his consistent support, guidance and clarifications during the running of this thesis.

I would also like to thank in advance Pr. El Madjid Berkouk and Pr. Chakir Messaoud , for evaluating my project.

Contents

List of Figures

List of Abbreviations

KF	*Kalman Filter*
PDF	*Probability Density Function*
EKF	*Extended Kalman Filter*
LG EKF	*Lie Group Extended Kalman Filter*
PF	*Particle Filter*
SO(3)	*Special Orthogonal Group*
SE(3)	*Special Euclidean Group*
AUV	*Autonomous Underwater Vehicle*
ROV	*Remotely Operated Vehicle*

Part I

Part 1: Theory

Chapter 1

General Introduction

Lie theory,is a fundamental area of mathematics emerged and inspired from the idea of continuous transformations at the end of the $19^{th} century$. After many years, its impact has expanded to other technical and scientific sectors. More particularly, the robotics community has made a considerable effort in recent years to correctly describe estimate issues[1], where the need of reliability and precision are in high demand. Accurate state and measurement modeling is achieved by representing the dynamical system as an element in "Manifold" ,which, are smooth topologic surfaces of the Lie groups on which the state representations evolve, a simple example of this is representing the attitude of the system as an SO(3) element, providing us with a non-singular unique representation of the body's attitude contrary to the classical representations such as Euler angles representation or quaternion representation. Furthermore, Once the estimation has been properly described on the lie group of the dynamical system, a control scheme which drives the system to the desired state on the lie group is required. And so a generalization of the classical control methods such as Proportional derivative, Sliding mode, LQR control is vital [2].

In This work, we overcome the difficulty of designing controllers and observers on lie groups, delivering accurate estimates and controlling the system to the desired state, by introducing the lie group version of the kalman filter and the particle filter, and by Discussing controllers that evolve on the lie group, alongside with a practical implementation of observer-based controller on autonomous underwater vehicle on SE(2)

This report consists of four chapters. The first, giving general introduction about the Lie theory and providing explanation about how other calculus ideas can be extended to the manifold. The second chapter, delves deeper into how can control methods apply in systems evolving on lie groups such as SO(2), SO(3), $SO(3)X\mathbb{R}^3$. PD,Sliding mode and LQR control on lie groups are discussed alongside with numerical simulation results. The third chapter, tackles the problem of estimation on lie groups, and provides the lie version of extended kalman filter and particle filter, an example simulation is given with results and their discussion . The fourth chapter, we try to apply the developed ideas on a concrete robotic system, the chosen application is autonomous underwater vehicle ,AUV, and finally implementing An LG-EKF observer-based PD-controller on an autonomous underwater vehicle evolving on SE(2).

Chapter 2

The theory of lie groups

2.1 Introduction :

Lie theory, the theory of Lie groups, and lie algebra are an essential part of mathematics, which date back to the 19 century, it has been investigated by the mathematician Sophus Lie, who laid the foundation of continuous transformation groups. Many years later, its effect has spread across various fields in science and engineering. Recently,We've seen a significant increase in its use in robotics, in control but also in estimation which require the manipulation and estimation of 3D geometry and especially of euclidean transformation matrices .This has pushed the robotics community to properly characterize estimation problems. Which necessitate solutions that are precise, consistent, and stable. Indeed, fulfilling these objectives necessitates accurate modeling of states and measurements, as well as the functions that connect them and their uncertainty. This has resulted in designs which involve mathematical objects which are called **manifolds** which are smooth topological surfaces. We may build a rigorous calculus based on the lie theory to handle these uncertainties, derivatives and integrals without losing any precision whatsoever. These efforts have been concentrated on the well-known two manifolds which are SO(3) and SE(3)
Lie groups, on the other hand, are by no means simple and represent very abstract creations for the vast majority of roboticists, making them difficult to understand and employ.

In contrast,in lie theory for robotics it is frequently not necessary to use the theory to its full potential, and therefore an effort of selection of materials is required which is the goal of this first chapter. The recent research on Lie groups provides a handful introduction to lie group theory such as [1] ,[3] and also [4], [5], [6] which provide a good introduction to the topic. In this chapter we will discuss the necessary tools that the lie group theory provides and its powerful utility in expressing rigid body rotations SO(3), and rigid body motion SE(3), we will also elaborate more on the tangent space related to the Lie group, along with the relationship between them, calculus and uncertainty are also expressed in this chapter

2.2 Lie groups :

Before we define what a Lie Group is we should first define two mathematical concepts which are groups and manifolds
Group : Mathematicians invented the concept of a group to capture the essence of symmetry. The collection of symmetries of any object is a group, and every group is the symmetries of some object.More formally, a group is a set equiped with an operation or algebraic structure where every element whithin it obeys a certain rules which are :

Intern Law :$a, b \in G \to a * b \in G$
Associativity : $a, b, c \in G \to (a * b) * c = a * (b * c)$
Identity element : $\exists e \in G$ tel que pour a $\in$ G a*e = e*a = a
Inverse element : $\forall a \in$ G $\exists$ un b tel que $a * b = b * a = e$

Lie groups:
Lie groups are elements which perserve the structure of a group, but are also a differentiable manifold, a manifold means that it is a topological space which locally resembles the eucledian space near each point
Action of a Lie group : Importantly, Lie groups have the potential to transform elements of other sets, producing e.g. rotations, translations, scalings, and combinations of them. These are extensively used in robotics, both in 2D and 3D. The formal definition of action of a lie group is as follows :
we define the action of a group G on another set V if it satisfies the next properties :

- identity : $e.v = v$
- Compatibility: $(g * g').v = g.(g'.v)$

2.3 Examples of Lie groups :

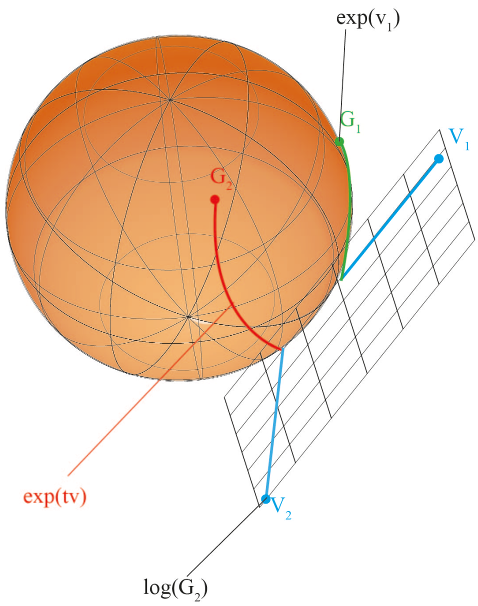


Figure 2.1: a geometric view of a lie group and its tangent space "Lie algebra"

Common examples of Lie groups include Rotation matrices SO(3) and the group of rigid motion SE(3) which will be cited here :

2.3.1 Complex numbers :

Our first example of Lie group, which is the easiest to visualize, is the group of unit complex numbers under complex multiplication with the form : $\mathbf{z} = cos(\theta) + isin(\theta)$ which is in fact a group which could be verified by the proporties cited above
Action : the group action is that it rotates other complex numbers by an angle θ

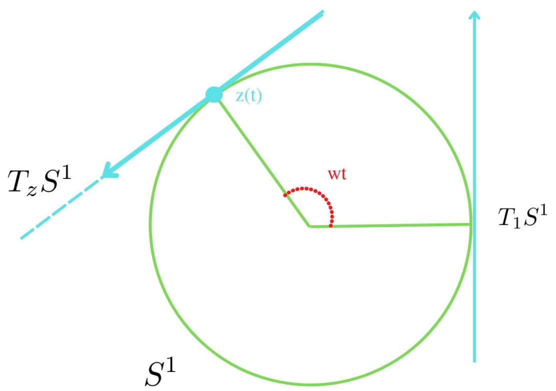


Figure 2.2: complex numbers Lie group

2.3.2 Special Orthogonal groups SO(3) :

the Special Orthogonal group is defined as :

$$SO(3) = \{R|R^T * R = I\} \tag{2.1}$$

However, SO(3) is not a valid vectorspace, despite the fact that the set of all matrices may be demonstrated to be a vectorspace. Because SO(3) is not closed under addition, adding two rotation matrices does not yield a valid rotation matrix:

and

$$R_1 + R_2 \notin SO(3) \tag{2.2}$$

This set however can be proven to have the structure of a Lie group, which again proves the utility of the theory.

2.3.3 Special Eucledian group SE(3):

Another important example in robotics is the set of rigid motion transformation which is defined as :

$$SE(3) = \{\begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \mid R \in SO(3)\} \tag{2.3}$$

This group presents the rigid body motion since it presents both the attitude and the position of the system at the same time, this description proves to be very useful in robotics.

2.4 Lie Algebra :

To be able to visualize the idea of Lie Algebra, one has to regard the concept of a manifold as a smooth surface or hyper-surface with no edges or spikes in it , In robotics, we say that our state vector evolves on this surface meaning that the surface is determined by the state constraints. From this, one can deduce that since the lie group is manifold which looks the same at each point then the tangent space at any point is the same.A special tangent space is the tangent space at the identity point which is also called **Lie Algebra**. More formally, given $X(t)$ a point moving on a manifold M, its velocity $\dot{X} = \frac{\partial X}{\partial t}$ belongs to the tangent space at that point which we denote $T_X M$

Lie Algebra :

Lie algebras can be defined locally to a point X, establishing local coordinates for $T_X M$. We shall denote elements of the Lie algebras with a 'hat' decorator. To be more precise, a lie algebra is a vector space ,which means that there exists a vector space $\mathbb{R}^m$ with $m \in$ N to which the lie algebra is isomorphic to. This vector space has Lie Brackets, which is governed by a law, which must meet the following criterias:

Bilinearity :$[ax + by, z] = a[x, z] + b[y, z]et[x, by + cz] = b[x, y] + c * [x, z]$
alternativity : $[x, x] = 0$
Jacobian identity : $[x, [y, z]] + [z, [x, y]] + [y, [z, x]] = 0$
anti-commutativity: $[x, y] = -[y, x]$

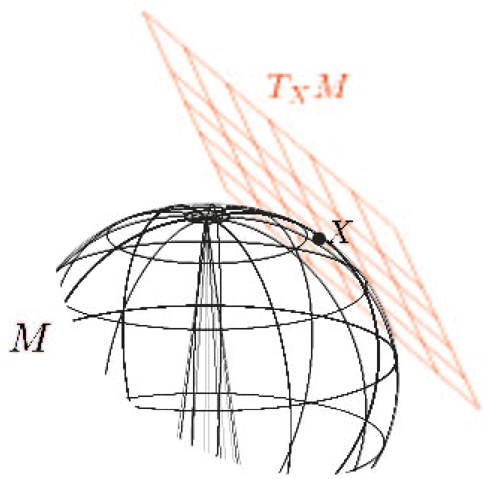


Figure 2.3: The lie algebra

The utility of a tangent space is that it is a vector space so that we can use all the linear algebra developed but also its potential is that it can fully represent the lie group.
The Lie Algebra's elements have nontrivial structures(Skew-Symmetric), but the most important property for us is that they can be written as linear combinations of some base elements and are isomorphic to $\mathbb{R}^n$. It is then convenient to manipulate only the coordinates as vectors in $\mathbb{R}^m$ which we will simply refer to as τ

$$Hat : \mathbb{R}^n \rightarrow m$$

$$Vee : m \rightarrow \mathbb{R}^n$$

Vectors in $\mathbb{R}^n$ are handier since they just require less memory to store which means less computaion and hence it is preferred in our simulation over the lie algebra m.
in the next sections we'll investigate the relationship between the Lie group and the Lie algebra.

2.5 Exponential Map:

The exponential map has to be thought intuitively as the lie algebra wrapping the lie group, and it allows us to precisely transfer elements of the Lie algebra to the group which wrap the tangent element around the manifold in a large arc or geodesic. The inverse operation of this is the $log()$,the unwrapping operation. more formally :

- m : is the lie algebra
- M : is the lie group

$$exp : m \rightarrow M$$
$$\log :\ \mathrm{M} \rightarrow m$$

The totally convergent Taylor series is used to obtain closed forms of the exponential in multiplicative groups.

$$\exp(\hat{\tau}) = \epsilon + \hat{\tau} + \tfrac{1}{2!}\hat{\tau}^2 + \tfrac{1}{3!}\hat{\tau}^3 + ...$$

From this, noting that this is the convenient exponential for matrices, one can deduce the following key properties of the exponential map drived from [1]:

$$\begin{cases} exp((t+s)\hat{\tau}) = exp(t\hat{\tau})exp(s\hat{\tau}) \\ exp(t\hat{\tau}) = exp(\hat{\tau})^t \\ exp(-\hat{\tau}) = exp(\hat{\tau})^{-1} \\ exp(X\hat{\tau}X^{-1}) = Xexp(\hat{\tau})X^{-1} \end{cases}$$

which represent powerfull properties which we will make use of later

2.5.1 The capitalized Exp and Log :

the capitalized Exp and Log are convenient shortcuts to map vector elements $\tau \in \mathbb{R}^n$ directly to elements $X \in M$ we have :

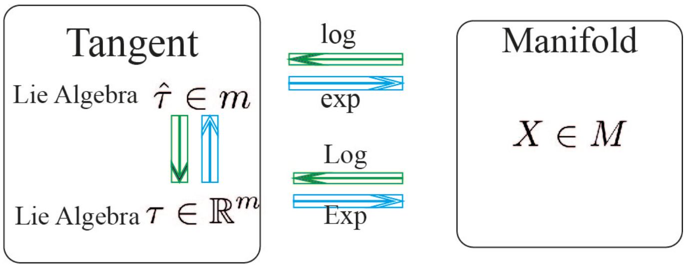


Figure 2.4: The relationship between the lie algebra and its lie group

$$\begin{array}{lll} \text{Exp}: & \mathbb{R}^m \rightarrow M; & \tau \rightarrow X = Exp(\tau) \\ \text{Log}: & M \rightarrow \mathbb{R}^m; & X \rightarrow \tau = Log(X) \end{array}$$

Note: It should be noted that the exponential operation is surjective-only meaning that we can obtain the same element $G \in M$ from different elements from the lie algebra $g \in m$, this property is important because it allows us to define the inverse mapping which is the logarithmic map. To analyse this more concretely we will use the following from [7]:

$$exp(\hat{\Phi}) = exp(\phi \hat{a}) \tag{2.4}$$

$$= 1 + \phi \hat{a} + \frac{1}{2!}\phi^2 \hat{a}\hat{a} + \frac{1}{3!}\phi^3 \hat{a}\hat{a}\hat{a} + ... \tag{2.5}$$

$$= aa^T + (\phi - \frac{1}{3!}\phi^3 + \frac{1}{5!}\phi^5) - (1 - \frac{1}{2!}\phi^2 + \frac{1}{4!}\phi^4) \tag{2.6}$$

$$= cos(\phi)\mathbf{1} + (1 - cos(\phi))aa^T + sin(\phi)\hat{a} = C \tag{2.7}$$

and since

$$\hat{a}\hat{a} = -1 + aa^T \tag{2.8}$$

$$\hat{a}\hat{a}\hat{a} = -\hat{a} \tag{2.9}$$

showing the last proposition that the exponential map is surjective only and so :

$$C = exp((\phi + 2\pi m)\hat{a}) \tag{2.10}$$

if we limit the angle of rotation to : $|\phi| < \pi$ we can define the logarithmic map as follows :

$$\phi = cos^{-1}(\frac{tr(C) - 1}{2}) + 2\pi n$$

we should also mention the Rodriguez formula for calculating the exponential mapping :

$$R = I + (sin(\theta)K) + (1 - cos(\theta))K^2 \tag{2.11}$$

2.6 Plus and Minus operators :

In order for us to define derivatives on matrix lie groups, we first have to inverstigate the operators plus and minus, first we define a plus operator which operates between an element X in the lie group and a vector in $\mathbb{R}^m$, intuitively speaking this operator when projected to the rotation matrices group it propagates the rotation according to the angular velocity (the lie algebra) while the minus operator represents the actual difference between the two matrices since the operation $R_2 - R_1$ doesn't signifie any meaning information about the closessness of those two matrices. These operators are used in the discrete case because they keep the new generated matrix in the same lie group and thus it is perserving the structure of the lie group

The plus and Minus operators: are defined as follows $X, Y \in M$ and $\tau \in \mathbb{R}^m$:

$$X \oplus \tau = X * exp(\hat{\tau}) \tag{2.12}$$

$$X \ominus Y = X^{-1} * Y \tag{2.13}$$

Note: It should be noted that there are two versions of this operators, the precedent being the left Plus-Minus Operator and the next one is the right Plus-Minus Operator :

$$X \oplus \tau = Exp(^{\varepsilon}\hat{\tau}) * X \tag{2.14}$$

$$X \ominus Y = Y * X^{-1} \tag{2.15}$$

It must also be noted that the left operator is expressed in the global frame(the lie algebra) while the right operator is expressed in the local frame (the tangent space at that perticular point $X \in M$), the link between the global and the local tangent spaces and the relationship between the two operations is what leads us to the concept of **the Adjoint** .

We will use the adjoint matrix often as a way to linearly transform vectors of the tangent space at X onto vectors of the tangent space at the origin, Ad_x, and the adjoint matrix is the following :

$$Ad_x(\tau) = X \tau X^{-1} \tag{2.16}$$

A more elaboration on adjoints can be found on the [1]

2.7 Derivatives and integrals on Lie groups :

We focus on derivatives defined as Jacobian matrices translating vector tangent spaces among the various approaches to express derivatives in the context of Lie groups. This is sufficient in this case because uncertainties and increments can be accurately and simply defined in these spaces. The formulas for uncertainty management in Lie groups will resemble those in vector spaces if these Jacobians are used.The derivative form is derived from the classical definition of a derivitive with a slight difference in the plus and minus operators:

$$\frac{^{X}Df(X)}{DX} = \lim_{h \to 0} \frac{f(x \oplus h) \ominus f(x)}{h} \tag{2.17}$$

which can be simplified to the following form :

$$\frac{{}^{X}Df(X)}{DX} = \lim_{\tau \to 0} \frac{\partial Log(f(X)^{-1} * f(x * Exp(\tau))}{\partial \tau} \tag{2.18}$$

This definition obviously doesn't seem intuitive at first, and so an example is provided :
Example : Calculating the jacobian of this simple function [1]

$$f : SO(3) \to R^3 \tag{2.19}$$

$$f(R) = Rp \tag{2.20}$$

$$\frac{{}^{X}Df(X)}{DX} = \lim_{\theta \to 0} \frac{f(x \oplus \theta) \ominus f(x)}{\theta}$$

$$\frac{{}^{X}Df(p)}{DX} = \lim_{\theta \to 0} \frac{(R \oplus \theta)p \ominus Rp}{\theta}$$

$$\frac{{}^{X}Df(p)}{DX} = \lim_{\theta \to 0} \frac{(RExo(\theta))p - Rp}{\theta}$$

$$\frac{{}^{X}Df(p)}{DX} = \lim_{\theta \to 0} \frac{(R(I + [\theta]_x)p - Rp}{\theta}$$

$$\frac{{}^{X}Df(p)}{DX} = \lim_{\theta \to 0} \frac{R[\theta]_x p}{\theta}$$

and taking into consideration that :

$$[a]_x b = -[b]_x a \quad if a, b \in \mathbb{R}^3 \tag{2.21}$$

from this we can conclude that :

$$\frac{{}^{X}Df(p)}{DX} = \lim_{\theta \to 0} \frac{-R[p]_x \theta}{\theta}$$

and so :

$$\frac{{}^{X}Df(p)}{DX} = -R[p]_x$$

And if we were to choose the right plus-minus one can make of the next property:

$$\frac{{}^{\varepsilon}Df(X)}{DX} Ad_X = Ad_{f(X)} \frac{{}^{X}Df(X)}{DX} \tag{2.22}$$

Discrete integration in manifolds :
The continuous-time integral of constant velocities $v \in T_{X_0}M$ onto the manifold is performed by the exponential map $X(t) = X_0 Exp(vt)$. Non-constant velocities v(t) are usually segmented into piecewise constant bits $v \in T_{X_{k-1}}M$, of short length δt_k, and the discrete integral is written. as

$$X_k = X_0 \circ Exp(v_1 \delta t_2) \circ Exp(v_2 \delta t_3).... \tag{2.23}$$

or it can be obtained recursively using :

$$X_k = X_{k-1} \circ Exp(v_k \delta t_k) \tag{2.24}$$

This recursive integration method is proved to keep the next k state within the manifold. This integration method will be used in discrete simulation via Matlab.

2.8 Uncertainty

In robotics, many estimation algorithms such as Kalman filter or Particle filter use probability concepts which are based on uncertainty in the vectorspace $\mathbb{R}^n$. To build an equivalent structure on the Matrix Lie group, a rigorous mathematical framework should be built .
Local perturbations at a certain point $X \in M$ in the tangent space $T_X M$ using Right $\oplus$ and $\ominus$:

$$\tau = \bar{X} \ominus X \in T_{\bar{X}} M \tag{2.25}$$

Defining the perturbation around a point on the lie group allows us to define covariance matrices which can be properly defined through the expectation operator $E[.]$

$$E(\tau\tau^T) = E((X \ominus \bar{X})(X \ominus \bar{X})^T) \in \mathbb{R}^{m*n} \tag{2.26}$$

defining covariances allows us to define gaussian distribution on Lie groups

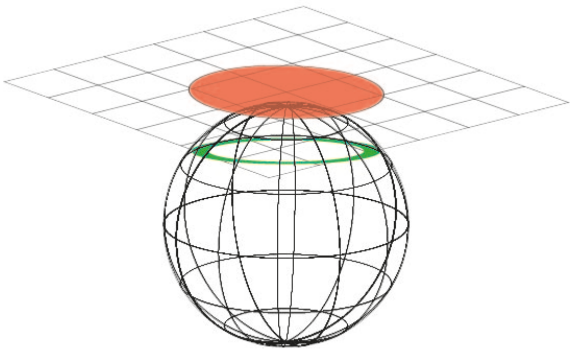


Figure 2.5: uncertainty of lie group elements

2.9 Conclusion :

Lie theory is a complicated subject, however, in robotics, it is frequently the case that we only need to explore a small subset of material that covers the essential parts of the theory which we can use later on. This chapter provides exactly this, by introducing what lie groups are ,and what are the the most common examples, and by developing calculus and probability concepts on lie groups. This helps us to be equipped for the next chapter.

Chapter 3

Control Design On Lie groups :

3.1 Proportional Derivative Control on the Eucledean group :

3.1.1 Introduction :

In this section, we will study the stabilization problem for control systems which evolve on the $SE(3)$ and $SO(3)$ Lie groups based on lyapunov stability in the aim of generalizing the classical Propotional derivative (PD) Control to the lie group Version. The analysis presented here will focus on systems of first and second order with one actuator for each degree of freedom, the geometric properties of Lie groups and lie algebras is used to generalize the classical approach. The compactness of the lie groups SO(3) and SE(3) (meaning that the space that they exist whithin have no holes) results in a natural metric structure. However, the definition of a metric is not unique which gives more freedom in the control design.
We here consider the problem of controlling a mechanical system whose configuration space is a matrix Lie group, an analysis of the stabilization problem is provided for the fully actuated case, in the hope of yielding a more advantageous results by using the geometric approach. Recent research shows that this is the case and that control schemes that use Lie theory outperforms conventional control methods in terms of response time and complex maneuver behaviors such as in [8]. [9] shows a robust attitude controller based on SO(3).
In this chapter we will introduce classical control methods on Matrix lie group for SO(3), SO(2). Such as [10] and for sliding mode, [11], but most importantly the following article which subsliding surfaces and control on them [2]for LQR :[12], [13], another general purpose control articles has also been useful for the derivation of the following such as: [14],[15] [16],[17]

3.1.2 First order systems and second Order systems on SO(3) :

We have to construct the mathematical background behind the metric concepts on Lie groups :
Metric properties on compact Lie groups :
On any Lie group C, the Killing form $\langle X, Y\rangle_k$ is defined as the bilinear operator on gg:

$$\langle X, Y\rangle_k = tr(ad_X, ad_Y) \quad \forall X, Y \in g. \tag{3.1}$$

An inner product on the Lie algebra g is defined based on the Killing form,above so that it satisfies the property of Ad-invariance which is:

$$\langle X, Y\rangle = \langle ad_g X, ad_g Y\rangle \quad \forall g \in G. \tag{3.2}$$

This gives the additional structure of a Riemannian manifold to the group G, Although this ad-invariance property doesn't seem intuitively useful at first but it is fundemantal to adress the following propositions which will be used to analyse the stability of the 1^{st} and 2^{nd} order systems

Proposition 1[4] :

Regarding the Ad-invariant metric, the distance between the element g and the identity $e_G = I \in G$ is given by the norm of the logarithmic function :

$$||g||_G = \langle log(g), log(g) \rangle^{\frac{1}{2}} \tag{3.3}$$

Theorem [4]:

Let G be a compact Lie group

with bi-invariant metric $\langle ., . \rangle$. Consider a smooth trajectory $g(t) \in G$, such

that $g(t)$ never passes through a singularity of the exponential map. then :

$$\frac{d}{dt}||g||_G^2 = \langle log(g), V^b \rangle = \langle log(g), V^s \rangle \tag{3.4}$$

where V^s and V^b are the angular velocity described in the local and global frame respectively
Once these two theorems are established, We start our analysis for stabilization of systems which evolve on a compact, semi-simple Lie group. We will focus our attention first on systems which evolve on $SO(3)$ described as in the equation :

$$\dot{R} = RV^b \tag{3.5}$$

Consider the natural candidate Lyapunov function

$$W(g) = \frac{1}{2}||g||_{SO(3)}^2 \tag{3.6}$$

V_b is our control quantity which should be within the lie algebra $so(3)$, this allows us to define a proportional control law similar to the case of classical proportional control :

$$V^b = -k_p log(g), k_p > 0 \tag{3.7}$$

which leads to

$$\dot{W}(g(t)) = \langle log(g), -k_p log(g) \rangle = -2k_p W \tag{3.8}$$

by using the lyapunov second theorem one can conclude that the exponential stability is assured for all initial conditions g_0, a simulation of this system is provided after the analysis of the second order system .

3.1.3 Second Order systems :

The first order system utilizes the speed as the control input and not the torques and forces, while dynamical systems and standard control problems such as problems in robotics and mechanics require a representation which is more rich to consider the dynamics of the system and to include the forces which is the goal of representation of second order systems which has the next form :

$$\begin{cases} \dot{R} = RV^b \\ \dot{V^b} = f(g, V^b) + U \end{cases}$$

The system is fully actuated by $U \in so(3)$ where $f(g, V^b) \in so(3)$ is the internal drift which depend often on the rotational inertia.
, A Proportional Derivative control can be used on the Lie group, and so in addition to the last control input used in V^b a derivative term must be added which is proportional to the velocity V^b
Theorem 2: [4]
Given the second order system equations, and let K_p and K_d be symmetric, positive definite gains. then the control law :

$$U = -f(g, V^b) - K_p log(g) - K_d V^b \tag{3.9}$$

exponentially stabilizes the state g at $I \in SO(3)$ taking into consideration the initial conditions in the following equation :

$$\lambda_{min}(K_p) > \frac{||V^b(0)||^2}{\pi^2 - ||g(0)||^2_{SO(3)}} \tag{3.10}$$

where $\lambda_{min}(K_p)$ is the minimus eigenvalue of K_p

Proof :
Taking the next Lyapunov Candidate :

$$W_e = \frac{1}{2}\langle \begin{bmatrix} log(g) \\ V^b \end{bmatrix}, \begin{bmatrix} id_{SO(3)} & \epsilon id_{SO(3)} \\ \epsilon id_{SO(3)} & K_p^{-1} \end{bmatrix} \begin{bmatrix} log(g) \\ V^b \end{bmatrix} \rangle_{SO(3)SO(3)} \tag{3.11}$$

The closed loop system satisfies the next equations :

$$\begin{cases} \dot{R} = RV^b \\ \dot{V^b} = -K_p log(g) - K_d V^b \end{cases}$$

substituting the $X = log(g) \in so(3)$, we obtain

$$\begin{cases} \dot{X} = \sum_{n=0}^{\infty} \frac{(-1)^n B_n}{n!} ad_X^n(V^b) = B_X V \\ \dot{V} = -K_p X - K_d V \end{cases}$$

Differentiating the lyapunov function gives:

$$\frac{d}{dt} W_e = \langle X, B_x V\rangle + \langle V, K_p^{-1}\dot{V}\rangle + \epsilon\langle B_x V, V\rangle + \epsilon\langle X, \dot{V}\rangle \tag{3.12}$$

$$= \langle X, V\rangle + \langle V, K_p^{-1}(-K_p X - K_d V)\rangle + \epsilon\langle B_x V, V\rangle + \epsilon\langle X, -K_p X - K_d V\rangle \tag{3.13}$$

= $-\epsilon\langle X, K_pX\rangle - \langle V, K_p^{-1}K_dV\rangle - \epsilon\langle X, K_dV\rangle + \epsilon\langle B_xV, V\rangle$(3.14)The last term can be upper bounded by $\epsilon\langle V, V\rangle$ using lemma 11 in [10] and hence the proposition becomes :

$$\frac{d}{dt}W_\epsilon \leq -\frac{1}{2}\langle \begin{bmatrix} X \\ V \end{bmatrix}, Q_\epsilon \begin{bmatrix} X \\ V \end{bmatrix}\rangle_{SO(3)SO(3)} \tag{3.15}$$

where

$$Q_\epsilon = \begin{bmatrix} \epsilon K_p & \epsilon\frac{K_d}{2} \\ \epsilon\frac{K_d}{2} & K_p^{-1}K_d - \epsilon id_{SO(3)} \end{bmatrix} \tag{3.16}$$

therefore local exponential stability is proven.
we'll show that the condition cited above about the minimum eigenvalue provides a sufficient condition to avoid singularities in the logarithmic map .

$$W(g) = \frac{1}{2}||g(t)||^2_{SO(3)} \leq W_0(t) \leq W_0(0) \tag{3.17}$$

$$= \frac{1}{2}||g(0)||^2_{SO(3)} + \langle V^b(0), K_p^{-1}V^b(0)\rangle_{SO(3)} \tag{3.18}$$

$$\frac{1}{2}||g(0)||^2_{SO(3)} + \lambda_{max}(K_p^{-1})||V^b(0)||^2 \tag{3.19}$$

$$\frac{1}{2}||g(0)||^2_{SO(3)} + \frac{1}{\lambda_{min}(K_p)||V^b(0)||^2} < \frac{\pi^2}{2} \tag{3.20}$$

which means that g(t) can never reach π and hence the logarithmic function can never be sigular completing the proof

3.1.4 Algorithms and simulation :

First Order system A library containing the operations discussed in Chapter 1 in lie groups has been built for SO(3),SO(2),SE(3),SE(2) via matlab :

Algorithm 1 Control of a first order system on SO(3) :

Data: u, θ_d, w_d, X_des
Result: $\hat{X}_k$
$dt \leftarrow 0.1$
$N \leftarrow 500$
$k_p \leftarrow 0.1$
initializing the first orientation $X_0 = eye(3)$
while *iteration* $<= N$ **do**
  $X \leftarrow exp^(\hat{u}) * X;$
  $u \leftarrow -dt * k_p * log(X_des' * X)$
  $k \leftarrow k + 1$

Results : The numerical simulation via Matlab is shown here

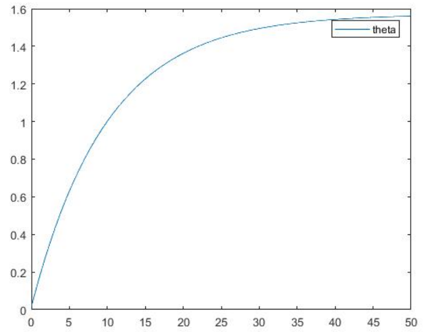


Figure 3.1: Angle :θ

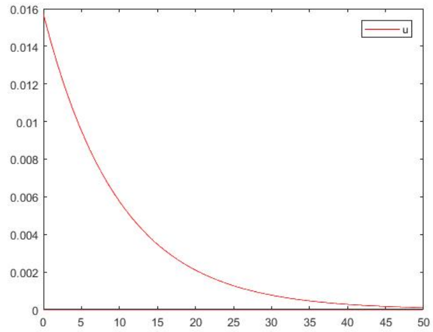


Figure 3.2: Control effort u

Second Order system In the first order system, the desired orientation is sufficient to design the controller, however this is not the case for the second order system, which needs both desired orientation and desired angular velocity:

Algorithm 2 Control of a Second order system on SO(3) :

Data: $u, \theta_d, w_d, X_des, w_d, J$
Result: $\hat{X}_k$
$dt \leftarrow 0.1$
$N \leftarrow 500$
$k_p \leftarrow 0.2$
$k_d \leftarrow 0.3$
$X_0 = eye(3)$
$w_0 = [0, 0, 0]$ initializing the first orientation and angular velocity
while $iteration <= N$ **do**
  $R \leftarrow exp^(\hat{u}) * R$
  $w \leftarrow J^{-1}((Jw)w + u)$
  $R_e = R_d' * R$
  $w_e = w - R_e' * w_d$
  $u \leftarrow -(Jw)w + J(-k_p log(R_e) - k_d w_e)$
  $k \leftarrow k + 1$

Results : The numerical simulation via Matlab is shown here

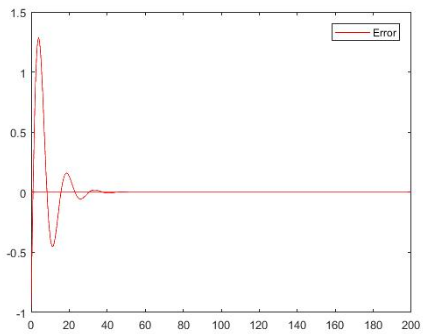


Figure 3.3: Theta Error

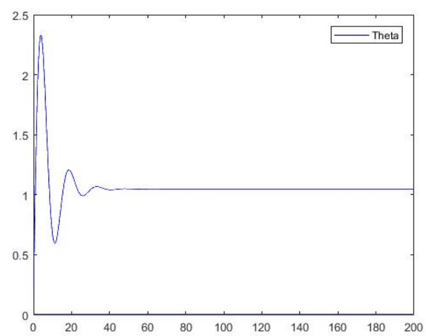


Figure 3.4: Angle θ

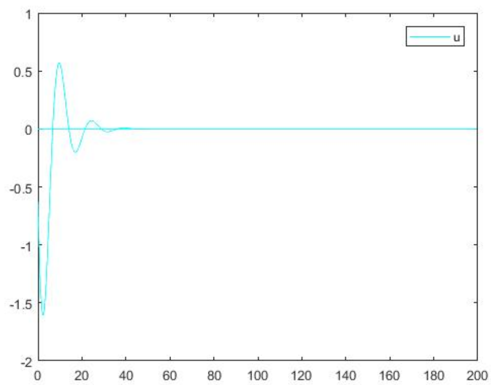


Figure 3.5: the control effort U

3.2 Sliding Mode control on SO(3) :

Sliding mode control is an efficient and robust tool to cast for the uncertainties in the model of the plant and to reject disturbances which are known feature that has been extensively documented in the literature. The design of such a controller is well understood and it is based on constraining the trajectories of the system to a specified linear surface which would reduce the order of the system and granting it to exist on the desired surface by making it attractive. In contrast, Manifold are, in mathematics, a generalization of the notion of a curved surface, so one can think that this notion can be easily extented to systems which exist on Lie groups.
There are some systems which do not evolve naturally on the euclidean space. It is possible that the state space isn't linear but have another key algebraic features. SO(3) which is discussed earlier, representing the attitude of rigid bodies, among other representations such as Euler angles, quaternions, and rotational matrices. Singularities can be found in euler angle parametrization, that is, points where the Jacobian of the coordinate chart loses rank, another inconvenience is that euler angles are not unique which leads to the unwinding phenomenon. Quaternion representation for the atittude are global with no singularity but they are still non-unique which makes SO(3) matrices a very interesting example. In such a case, it is necessary to create a more sliding surface which perserves the Lie group structure, that is a sliding surface that is a Lie subgroup rather than just a linear subspace. In this section a sliding surface is suggested.

3.2.1 Atittude dynamics :

The state space M of the attitude dynamics consists of the set of possible attitudes, SO(3), together with all possible angular velocities, $\mathbb{R}^3$, so that $M = SO(3)$ X R^3, we suppose that the system has a resisting inertia which we denote as $J = J^T > 0$
definition : Here we refer to w^x as equivalent to the hat operator on w and so $w^x = \hat{w}$

$$\begin{cases} \dot{R} = Rw^x \\ J\dot{w} = (Jw)w + u + d \end{cases}$$

where $u, d \in \mathbb{R}^3$ are the control and disturbance torques.
the dynamics of the desired attitude is :

$$\begin{cases} \dot{R}_d = R_d w_d^x \\ w_e = w - R_e^T w_d \end{cases}$$

the attitude and angular velocity error are as follows :

$$\begin{cases} R_e = R_d^T R \\ w_e = w - R_e^T w_d \end{cases}$$

Dynamics of the error :

$$\dot{R}_e = R_e w_e^x$$
$$J\dot{w}_e = J(\dot{w} - \dot{R}_e^T w_d - R_e^T \dot{w}_d)$$
$$J\dot{w}_e = (Jw)w + JR_e^T((R_e w_e)^x w_d - \dot{w}_d) + u + d$$

and hence the control law can be extracted as follows :

$$u = -JR_e^T((R_e w_e)^x w_d - \dot{w}_d) \tag{3.21}$$

which results in the following system :

$$J\dot{w}_e = (Jw)w + u + d \tag{3.22}$$

Stability analysis : A refinement of the LaSalle invariance principle for systems specified on manifolds will be used to prove almost global stability, we can refer to [10]. we can conclude that the system is almost global asymptotically stable.

3.3 Sliding-Mode control on $SO(3)X\mathbb{R}^3$:

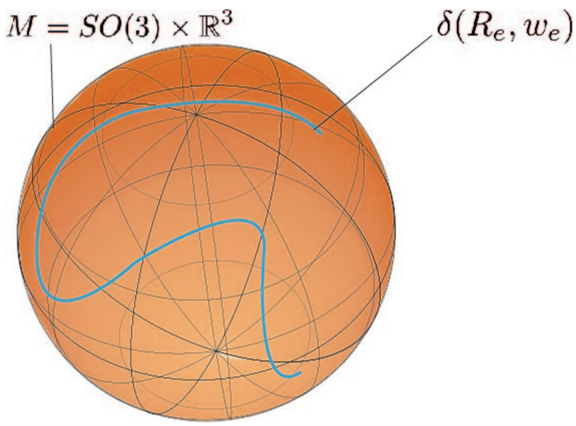


Figure 3.6: The sliding surface

First of all we should prove that $SO(3)X\mathbb{R}^3$ is a lie group provided the following operations:

$$(R_1, w_1).(R_2, w_2) = (R_3, w_3) \tag{3.23}$$

are defined as follows :

$$R_3 = R_1R_2 \tag{3.24}$$
$$w_3^x = P_a(R_1w_2^x + R_2^Tw_1^x - 1/2[R_1, R_2^T])$$

where $P_a(R) = 1/2(R - R^T)$ is the projection on the anti-symmetric matrices.
The proof that this is in fact a Lie group is done simply by verifying the group axioms since M is already a smooth manifold

The sliding mode surface :
The sliding surface is introduced in the following proposition. Consider the sliding variable $\delta : M \to \mathbb{R}^3$ this sliding variable is extracted from [5] and is defined as :

$$\delta(R_e, w_e) = w_e + vee(P_a(R_e)) \tag{3.25}$$

where vee stands for :

$$vee : \quad m \to \mathbb{R}^m$$

and the surface is the set which contains the following points

$$D = (R_e, w_e) \in M|\delta(R_e, w_e) = 0 \tag{3.26}$$

Proof : The smoothness of D is inherited from the smothness of both P_a and vee operator. We now verify the group axioms.

$$D = \delta(I, 0) = 0 + vee(P_a(I)) = 0 \tag{3.27}$$

D is closed under multiplication , and it can be verified using the lie bracket properties
D is closed under inversion :

$$\delta(R^T, -w) = -w^x + P_a(R^T) = -w^x - P_a(R) = 0 \tag{3.28}$$

Stability of the Reduced-Oder System : When we constrain the system to the constraint in the surface $\delta(R_e, w_e) = 0$, we obtain the reduced-Order system :

$$\dot{R}_e = -R_eP_a(R_e) \tag{3.29}$$

The identity $R_e = I$ is an almost globally stable equilibrium.
Proof : We will use the lyapunov candidate developped in [2] which is :

$$V_R(R_e) = \frac{1}{2}tr(I - R_e) \tag{3.30}$$

This function is obviously a a lyapunov function since $V_R(R_e) \geq 0$ and so the time derivative of it is :

$$\dot{V}_R(R_e) = \frac{1}{2}tr(R_eP_a(R_e)) \tag{3.31}$$

$$\dot{V}_R(R_e) = \frac{1}{2}\langle R_e^T, P_a(R_e)\rangle \tag{3.32}$$

$$\dot{V}_R(R_e) = \frac{1}{2}\langle -P_a(R_e) + P_s(R_e), P_a(R_e)\rangle \tag{3.33}$$

$$\dot{V}_R(R_e) = -\frac{1}{2}\langle P_a(R_e), P_a(R_e)\rangle \tag{3.34}$$

which leads finally to :

$$\dot{V}_R(R_e) = -\frac{1}{2}\langle vee(P_a(R_e)), vee(P_a(R_e))\rangle \tag{3.35}$$

$$\dot{V}_R(R_e) = -||vee(P_a(R_e))||^2 \leq 0 \tag{3.36}$$

and hence proving the stability of $R_e = I$

The Reaching Law :

The control law which enforces the system to the desired surface $\delta(R_e, w_e) = 0$ is

$$v(R_e, w_e, w) = -K(w_e, w)\frac{\delta(R_e, w_e)}{||\delta(R_e, w_e)||} \tag{3.37}$$

with :

$$K(w_e, w) \geq ||J||_2||w||^2 + ||w_e|| + d + \delta \tag{3.38}$$

in this case we suppose the states are bounded.

Proof :

The lyapunov candidate function :

$$V_\delta(\delta) = \frac{1}{2}\delta^T J\delta \tag{3.39}$$

The time derivative :

$$\dot{V}_\delta(\delta) = \delta^T J(\dot{w}_e + vee(P_a(\dot{(R_e)}))) \tag{3.40}$$

$$\dot{V}_\delta(\delta) = \delta^T((Jw)w + vee(P_a(R_e w_e^x)) + d + v) \tag{3.41}$$

since :

$$||(Jw)w|| \leq ||J||_2||w||^2 \tag{3.42}$$

and since :

$$\langle v, w\rangle = \frac{1}{2}\langle v^x, w^x\rangle \tag{3.43}$$

we have :

$$||vee(P_a(R_e w_e^x))|| = ||w_e|| \tag{3.44}$$

and so we conclude that :

$$\dot{V}_\delta(\delta) \leq -||\delta||(K(w_e, w) - ||J||_2.||w||^2 - ||w_e|| - \bar{d} \tag{3.45}$$

and taking into consideration the previous condition on K we finally have :

$$\dot{V}_\delta(\delta) \leq -c\sqrt{\frac{V_\lambda(\lambda)}{\lambda_{max}(J)}} \tag{3.46}$$

which implies that the system using this control law converges in finite time to the desired state.

3.4 Algorithms and Simulation :

Algorithm 3 Control of a Second order system on SO(3) :

Data: $u_0, \theta_d, w_d, X_des, w_d, J$

Result: $\hat{X}_k$

$dt \leftarrow 0.1$

$N \leftarrow 500$

$K \leftarrow 50$

$X_0 = eye(3)$

$w_0 = [0, 0, 0]$ initializing the first orientation and angular velocity

while $iteration <= N$ **do**

$R \leftarrow exp^(\hat{u}) * R$

$w \leftarrow J^{-1}((Jw)Xw + u)$

$R_e = R_d' * R$

$w_e = w - R_e' * w_d$

if iteration $== [\frac{N}{2}]$: change desired vector to w $= [0; 0; -\frac{\pi}{2}]$

$S = w_e + vee(\frac{1}{2}R_e - R_e^T)$

$V = -K * S/norm(S)$

$u = -J * R_e^T * (R_e \hat{*} w_d * w_d + V)$

$k \leftarrow k + 1$

Results : The numerical simulation via Matlab is shown here

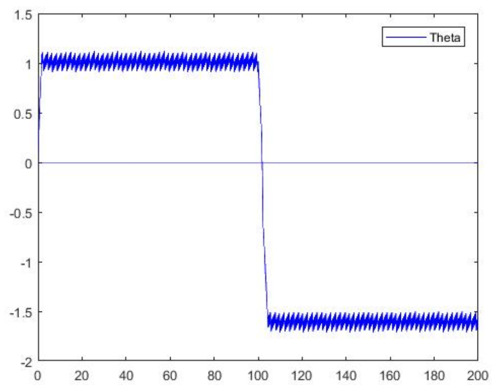


Figure 3.7: theta θ

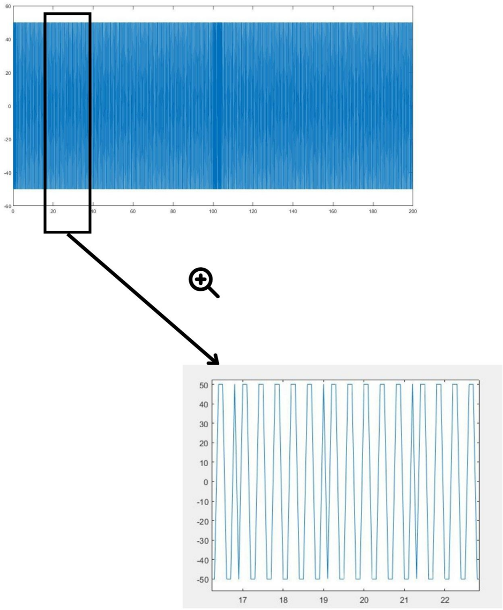


Figure 3.8: the control effort U

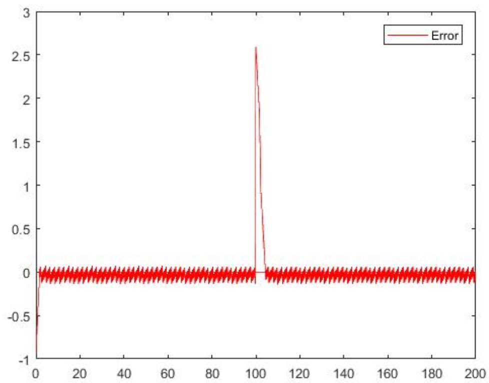


Figure 3.9: the error

3.4.1 Discussion :

It shall be noted that this sliding mode controller which is based on rotational matrices which are singularity free and unique.In contrast, designs which use quaternions suffer from being non-unique, while singularity still occurs in the euler angle representation, which makes designs that include rotation matrices better. This controller helped get rid of the unwinding phenomenon, and although this controller since it is based on the sliding mode concept then it has its own inconveniences as we can see the shattering phenomenon appearing in this case.

3.5 LQR Control on SO(2) :

3.5.1 Introduction :

Since we have explored the traditional Proportional derivativa Controller, alongside another robust controller such as sliding Mode control, the right next thing to explore is an optimal controller which is in this case LQR control on SO(2). In this section we are concerned with the optimal control scheme for controlling a linear system.
One of the oldest and most researched topics in control theory is the optimum regulation of a continuous time process.In this section an optimal linear controller is derived on the SO(2) manifold using a modelization of point mass system with rotational damping on the SO(2) manifold, formulated for optimal control, although the literature for analysing and discussing optimality in the lie Group is properly established and developped. Here we will discuss a practical and simple example as to show once again the powerfull extention to which Lie group theory can reach

3.5.2 System Dynamics :

Since we have established the Mathematical background for the Lie group theory for all SO(2), we'll model a basic SO(2)-defined system for optimum control. It is customary to modify the state x and the input u by substracting off a desired, trajectory x_{ref} ,
The point mass particle that will be used in this section is a particle constrained on the unit circle, its state on the circle can be described completely by two parameters which are : orientation θ and angular velocity w, based on [13] we will express the state as the following instead of regular mechanical modelization on $\mathbb{R}^3$:

$$x = \begin{bmatrix} \Phi \\ w \end{bmatrix}^T \quad u = \Gamma \tag{3.47}$$

Instead of θ we will express the orientation of the particle as the rotation matrix $\Phi \in SO(2)$ to avoid problems such as angle wrapping, and since Φ is not an element of a vectorspace, we will be using the operators on the lie group, therefore the difference between the desired state and the current state will be as follows :

$$\tilde{x} = x \ominus x_{des} = \begin{bmatrix} log(\Phi\Phi_{ref}^{-1}{}^{v} \\ w - w_{ref} \end{bmatrix} \tag{3.48}$$

where as discussed earlier the term $log(\Phi\Phi_{ref}^{-1})$ represents the difference between the desired and the actual state.
Dynamics of the error :
as it is shown here [13], it is quite obvious that

$$\dot{\Phi} = \hat{w} \in so(2) \tag{3.49}$$

$$\dot{\Phi} = \begin{bmatrix} 0 & -w \\ w & 0 \end{bmatrix} \in so(2) \tag{3.50}$$

and hence the state and its derivative is as follows :

$$\dot{x} = \begin{bmatrix} \dot{\tilde{\Phi}} \\ \dot{\tilde{w}} \end{bmatrix} = \begin{bmatrix} w \\ -\frac{b}{J}w + \frac{\Gamma}{J} \end{bmatrix} \tag{3.51}$$

the equations of the rotational acceleration are obtained by using Newton's second law to a rotating system with an angular velocity damping

3.5.3 LQR Control :

Based on [13],The LQR control method here operates on a dynamic control system by By minimizing an appropriate cost function. which is the following :

$$C = \frac{1}{2}\int_0^\infty x^T Q x + u^T R u \, dt \tag{3.52}$$

where $u = -Kx(t)$

$$K = R^{-1}B^T P \tag{3.53}$$

and P a matrix which satisfies :

$$-PA + A^T P + PBR^{-1}B^T P - Q = 0 \tag{3.54}$$

with Q and R are matrices which are positive definite

$$\begin{bmatrix} \dot{\tilde{\Phi}} \\ \dot{\tilde{w}} \end{bmatrix} = \begin{bmatrix} w - w_d \\ -\frac{b}{J}w + \frac{}{J} - (-\frac{b}{J}w_d + \frac{d}{J}) \end{bmatrix}$$

$$\begin{bmatrix} \tilde{w} \\ -\frac{b}{J}\tilde{w} + \frac{\tau}{J} - \frac{\tilde{\tau}}{J} \end{bmatrix} = f(x,u)$$

Linearizing the system by : A $= \frac{\partial f}{\partial x}$, B $=\frac{\partial f}{\partial u}$

3.6 Algorithm and Simulation :

K is derived from [13]

Algorithm 4 LQR Tracker on SO(2) :

Data: $Q, R, b, J, \theta_d, A, B, x_0, w_0, T_d, w_d$

Result: $\hat{X}_k$

$dt \leftarrow 0.1$

$N \leftarrow 500$

$K \leftarrow \begin{bmatrix} 10 & 5.378 \end{bmatrix}$

$X_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$w_0 = 0$ initializing the first orientation and angular velocity

while *iteration* $<= N$ **do**

$\Phi_d = exp(\hat{\theta}_d)$

$\tilde{x} = \begin{bmatrix} log(\Phi\Phi_d^{-1})^v \\ w - w_d \end{bmatrix}$

$\Gamma_1 = -K\tilde{x}$

$\Gamma = \Gamma_1 + \Gamma_d$

$\Phi(t + \Delta t) = exp(w(t)\Delta t)\Phi(t)$

$w(t + \Delta t) = w(t) + (\frac{1}{J}\Gamma - \frac{b}{J}w(t))\Delta t$

k$\leftarrow k + 1$

Results : The numerical simulation with matlab has provided the next results :

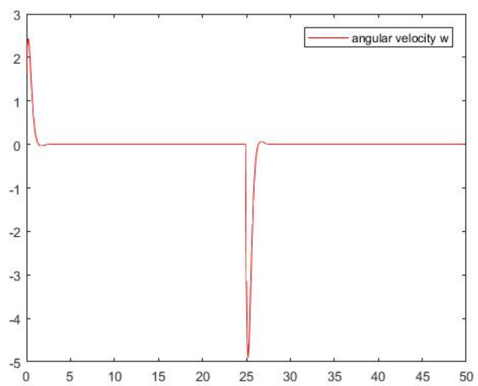


Figure 3.10: The angular velocity

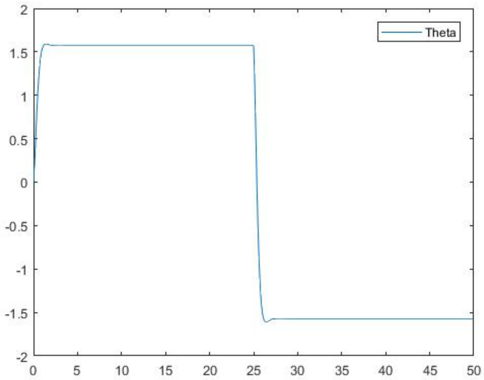


Figure 3.11: Theta θ

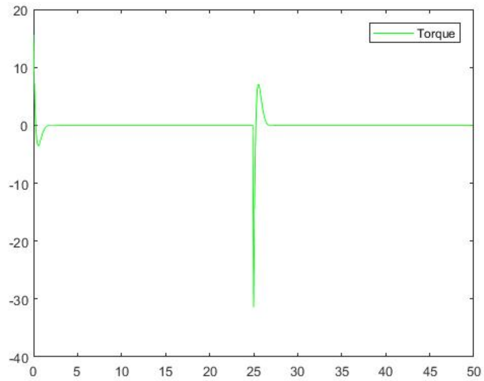


Figure 3.12: The Torque Γ

3.6.1 Conclusion :

Traditional control methods has the potential to be converted to their lie group version, this can be done by using various methods. In this chapter, we discussed precisely that by introducing three fundemantal classical methods the proportional derivative, sliding mode, and LQR control.

Chapter 4

State Estimation on Lie groups :

4.1 Introduction :

State estimate is useful in almost every field of engineering and research. Any subject concerned with mathematical modeling of its system is a plausible (if not unavoidable) candidate for state estimation.The sole limit to the uses of state estimation theory is the engineer's creativity, which is why it has become such an extensively investigated and applied topic in recent decades.
Estimation is the dual problem of control, and many robotics applications rely on the ability to estimate the system properly, such as pose estimation, object tracking, vision based estimation ...etc. because of the expanding usage of potentially fault, low-cost sensors, and an ever growing deployment of robotic algorithms in consumer products which operate in potentially unknown environments. It has become a necessity to design algorithms which can withstand strong non linearities, high uncertainty levels, and numerous outliers. However, particularly in robotics, the Gaussian assumption is widely spread across areas in reseach because of its applicable utility, the optimal problem for estimation is also taken into consideration which gave rise to the famous Kalman filter describing a recursive solution to the discrete-data linear filtering problem for linear systems, The Kalman filter has been extensively studied and applied, notably in the field of autonomous or assisted navigation, due to its efficiency both in performance and compuationally. Many variations of the Kalman filter has been introduced into the world of estimation such as the extended Kalman filter for Non linear systems, Unscented kalman filter a suboptimal non-linear filtration algorithm and many others. However, the kalman filter only treats estimation problems with possess errors that are gaussian distributed, this problem is solved with introducing another filter which is called the Particle filter. Although these estimation methods are useful, they have the disadvantage of only evolving on linear vector spaces, in this section we will treat all the algorithms cited above with the addition of discussing filters which evolve on Matrix Lie groups. This chapter is constructed with the help of the following [18] ,[19],[20],[21], [22], [23],[24], [25],while for particle filtering the following are used : [26],[27],[28],[29]

4.2 Kalman Filter :

The Kalman filter addresses the problem of estimating the state $x \in \mathbb{R}^n$ of a discrete time process which is governed by a linear stochastic difference equation

$$x_{k+1} = A_k x_k + B u_k + w_k \tag{4.1}$$

The measured outputs are also stochastic with a measurement $z \in \mathbb{R}^m$ with noise v_k

$$z_k = H_k x_k + v_k \tag{4.2}$$

the noises v_k and w_k are both process noise with the next normal distribution with covariance $E(w_k w_k') = Q$ and $E(v_k' v_k) = R$:

$$p(w) = N(0, Q)$$

$$p(v) = N(0, R)$$

the intuitive idea behind the Kalman Filter is that it estimates the state with optimal confidence between the measurements or the model of the state space. The kalman filter provides good estimation properties and is optimal in the special case when the process is linear and measurement follows a Gaussian distribution The filter evaluates the state of the process at a given point in time and subsequently receives feedback in the form of (noisy) measurements. This procedure makes the discrete filter a two step algorithm which start by propagating the state and the probability covariance at time x_k to the state at time step x_{k+1} using the model, this step is called prediction and the second step is correcting the estimated state given the measurements obtained by the noisy sensors in an optimal way which is called correction

4.2.1 Algorithm:

the Algorithm below covers both the steps denoted before

Algorithm 5 Kalman filter :

Data: A_k, B, U_k, v_k, w_k

Result: $\hat{X}_k$

$\hat{X_{k+1}}^{-} = A_k \hat{X}_k + Bu_k$

$\hat{P_{k+1}} = A_k \hat{P}_k A_k^T + Q_k$

$\mathrm{K}_k = P_k^{-1} H_k^T (H_k P_k^{-1} H_k^T + R_k)^{-}$

$\hat{X}_k = \hat{X_{k+1}}^{-} + K(z_k - H_k)\hat{X}_k^{-}$

$P_k = (I - K_k H_k) P_k^{-}$

4.3 Extended Kalman filter :

The Non Linear Process:

one of the inconveniences of a kalman filter, is that it only works with linear systems, but most real world applications are non-linear which make the kalman filter useless in such situations. However, such an inconvenience is tackeled by introducing the extended kalman filter which has produced one of the most successful applications .

In Extended Kalman Filter, we can linearize the estimation around the current estimate using the partial derivatives of the process and measurement functions to compute the estimates. This time our process is driven by the non-linear stochastic difference equation which has a state vector $x \in \mathbb{R}^n$.

$$x_{k+1} = f(x_k, u_k, w_k) \tag{4.3}$$

$$z_k = h(x_k, v_k) \tag{4.4}$$

First we need to linearize the system around the specified point in order to work with a system akin to the linear system discussed above :

$$A_{[i,j]} = \frac{\partial f}{\partial x}(\hat{x_k}, u_k, 0)$$

$$B = \frac{\partial f}{\partial u}(\hat{x_k}, u_k, 0)$$

$$W_{[i,j]} = \frac{\partial f}{\partial w}(\hat{x_k}, u_k, 0)$$

$$H_{[i,j]} = \frac{\partial h}{\partial x}(\hat{x_k}, 0)$$

$$V_{[i,j]} = \frac{\partial h}{\partial v}(\hat{x_k}, 0)$$

$$x_{k+1} = \tilde{x_{k+1}} + A(x_k - \hat{x_k} + W w_k$$

$$z_k = \tilde{z_k} + H(x_k - \tilde{x_k}) + V v_k$$

4.3.1 Algorithms :

Algorithm 6 Extended Kalman filter :

Data: A_k, B, U_k, v_k, w_k
Result: $\hat{X}_k$
$\hat{X_{k+1}}^- = f(x_k, u_k, w_k)$
$A_k = \frac{\partial f}{\partial x}(\hat{x_k}, u_k, 0)$
$H_k = \frac{\partial H}{\partial x}(\hat{x_k}, u_k, 0)$
$\hat{P_{k+1}} = A_k \hat{P}_k A_k^T + Q_k$
$\mathrm{K}_k = P_k^{-1} H_k^T (H_k P_k^{-1} H_k^T + R_k)^{-1}$
$\hat{X}_k = \hat{X_{k+1}}^- + K(z_k - H_k)\hat{X}_k^{-1}$
$P_k = (I - K_k H_k) P_k^-$

4.4 Extended Kalman filter on Matrix Lie group:

4.4.1 Introduction :

Throughout the previous chapters we have seen how elements on matrix lie group do not satisfy the basic operation that we take for granted. This theme continues when working with random variables, which are typically of the form

$$x \sim N(\mu, \sigma) \tag{4.5}$$

or an equivalent way to represent this is by spliting it into the central or the mean component and a noisy component which are :μ , ϵ respectively

$$x = \mu + \epsilon, \epsilon \sim N(0, \sigma) \tag{4.6}$$

And since all of the components are vectors which are closed under the + operation forming a vectorspace, this arrangement works. However, in Matrix Lie Groups, this is not the case since it is not closed under addition and so a new method should be developed in order to define random variables on Matrix Lie Group

4.4.2 Gaussian Random Variables and probability density functions On Lie Groups :

The first step to define random variables on a lie group is to exploit the vectorspace properties of the Lie Algebra, this way we can utilize all the developed concepts from probability and statistics rather than just building all the theory from scratch. Knowing this, we can define a variable on SO(3) for example as follows :

$$C = exp(\hat{\epsilon_l})\bar{C} \tag{4.7}$$

where ϵ is a random variable in the usual sense and $\bar{C} = exp(\Phi)$ and this simply means that a probability distribution function on $\mathbb{R}^3$ will induce a probability distribution function on $SO(3)$

$$p(\epsilon) \rightarrow p(C)$$

we will be mostly interested by the gaussian distribution because it is used for the kalman filter
the gaussian distribution has the next form for the multivariable case :

$$p(\epsilon) = \int \frac{1}{\sqrt{(2\pi^3)det(\sum)}} exp^{(-\frac{1}{2}\epsilon^T \Sigma^{-1} \epsilon)}$$

$p(\epsilon)$ is a valid PDF by definition and so :

$$\int p(\epsilon)d\epsilon = 1$$

referring to the first chapter, we have the next property:

$$dC = |det(J(\epsilon))|d\epsilon$$

this property leads to :

$$1 = \int p(\epsilon)d\epsilon$$

$$= \int \frac{1}{\sqrt{(2\pi^3)det(\sum)}} ex^{(-\frac{1}{2}\epsilon^T \Sigma^{-1} \epsilon)}d\epsilon$$

$$= \int \frac{1}{\sqrt{(2\pi^3)det(\sum)}} exp^{-\frac{1}{2}ln(C\bar{C}^T)^{vT} \Sigma^{-1} ln(C\bar{C}^T)^v} \frac{1}{|det(J)|} dC$$

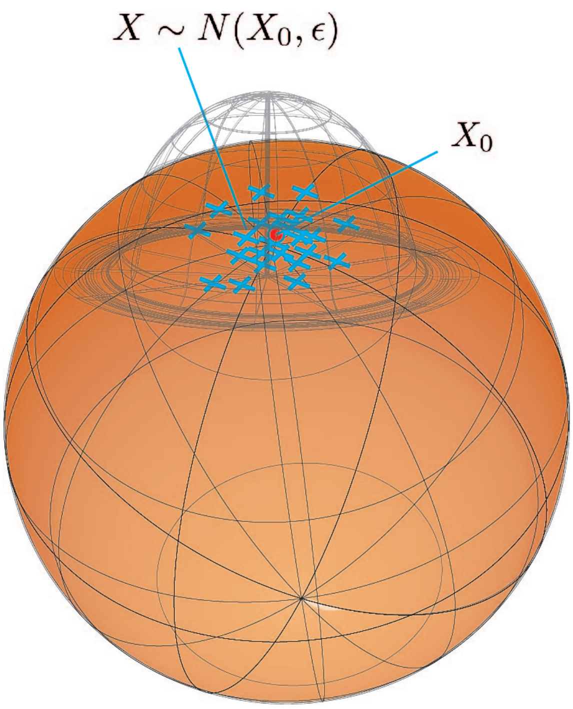


Figure 4.1: Probability Distribution on Lie groups

4.4.3 Discrete Extended kalman filter on lie groups :

We utilize the previously developed theory on gaussian distribution on Lie groups in oreder to develop the next LG-EKF :

4.4.4 System Model :

Since many systems utilize rotation matrices, we can think of the following model as a rotation matrix controlled by an angular velocity . More formally, the system evolves on a manifold satisfying the following equations :

$$X_k = X_{k-1} exp_G([\Omega(X_{k-1}, U_{k-1}) + n_{k-1}]_G)$$

where :

$$X_k \in G, u_{k-1} \in \mathbb{R}^m \tag{4.8}$$

n_{k-1} is white gaussian noise, Ω : a non linear function
the mesurement model also evloves on Lie group :

$$z_k = h(X_k) exp_{G'}([w_k]_{G'})$$

4.4.5 The D-LG-EKF :

As for the EKF, the D-LG-EKF also has two steps, the first is to propagate the state obtaining the posterior state , and the second is to update the distribution parameters, which are $\mu_{k-1|k-1}$ and $P_{k-1|k-1}$ where μ represents the mean and P is the covariance matrix.

4.4.6 Propagation :

The aim of this part is to show how to propagate $\mu_{k-1|k-1}$ and $P_{k-1|k-1}$ between two consecutive sensor readings.

4.4.7 mean Propagation :

the mean is updated according to the next equation :

$$\mu_{k|k-1} = \mu_{k-1|k-1} exp_G([\Omega]_G)$$

4.4.8 Covariance propagation :

Studying the Lie algebraic error propagation yields that the state error on G is expressed as follows:

$$exp_G([\epsilon_{k|k-1}]) = \mu_{k|k-1}^{-1} X_k$$
$$= exp(-[\Omega]_G) exp_G([\epsilon_{k-1|k-1}])$$

and given the properties cited in the first chapter this leads to the following equation :

$$= exp([\Omega(X_{k-1}, u_{k-1}) + n_{k-1}]$$

which results finally by using equations 1 and 2 to the following update :

$$\epsilon_{k|k-1} = F_{k-1}\epsilon_{k|k-1} + \Phi(\Omega_{k-1}) n_{k-1}$$

where :

$$F_{k-1} = ad(exp(-\Omega)) + \Phi(\Omega_{k-1})_{k-1}$$
$$Z_{k-1} = \frac{\partial}{\partial \epsilon} \Omega(\mu_{k-1|k-1} exp([\epsilon]), u_{k-1})_{|\epsilon=0}$$

completing the prediction step

4.4.9 Correction :

In this stage, the information coming from the measurement should be used to update or to correct the posterior estimate of the state X

$$\bar{z_k} = [log(h(\mu_{k|k-1})^{-1}) z_k]$$

$$\bar{z_k} = [log(exp(H_k \epsilon_{k|k-1}) exp([x_k])]$$

where

$$H_k = \frac{\partial}{\partial \epsilon} [log(h(\mu_{k|k-1})^{-1} h(\mu_{k|k-1} exp([\epsilon])))]$$

and

$$z_k = H_k \epsilon_{k|k-1} + w_k$$

we use classical update equations of the Kalman filter to update $\epsilon_k k-1$ giving the next steps :

$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + Q_k)^{-1}$
$m_{k|k}^- 1 = 0_{p1} + K_k(z_k - H_k)\ \mathbf{0}_p 1$
$\mathrm{P}_{k|k}^- = (Id - K_k H_k) P_{k|k-1}$
$m_{k|k} = 0_p 1$
$P_{k|k} = \Phi(m_{k|k}) P_{k|k} \Phi(m_{k|k})^T$

4.5 Algorithm and Simulation :

Algorithm 7 D-LG-EKF Algorithm :

Input : $\mu_{k-1|k-1}, P_{k-1|k-1}, U_{k-1}, z_k$
Output : $\mu_{k|k}, P_{k|k}$
Propagation
$\mu_{k|k-1} = \mu_{k-1|k-1} exp_G([\Omega]_G)$
$P_{k|k-1} = F_{k-1} P_{k-1|k-1} F_{k-1}^T + \Phi(\Omega_{k-1}) R_{k-1} \Phi(\hat{\Omega_{k-1}})^T$
$K_k = P_{_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + Q_k)^-}$
$m_{k|k}^- = K_k([Log(h(\mu_{k|k-1}^{-1} z_k))])$
$\mu_{k|k} = \mu_{k|k-1} exp([m_{k|k}])$
$P_{k|k} = \Phi(m_{k|k}^-)(Id_I - K_k H_k) P_{k|k} \Phi(m_{k|k}^-)$

4.5.1 Simulation :

The targeted problem here is Kalman filter for landmark-based localization for a robot in 2D.
We consider a rigid body or a robot in a 2D plane surrounded by a small number of punctual landmarks. We suppose that the robot has the capacity to measure the location of the landmarks with respect to its own reference frame. The robot has to estimate its state based on the noisy measurements of the landmarks, axial and angular velocities are the control actions that affect the robot. The pose of the robot is modeled as $SE(2)$ since the problem considered is in 2D and landmark positions are $b_k \in \mathbb{R}^2$

$$X = \begin{bmatrix} \mathrm{R} & \mathrm{t} \\ 0 & 1 \end{bmatrix} \in SE(2), \quad b_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix} \in \mathbb{R}^2 \tag{4.9}$$

The control input is made up from only longitudinal velocity v and angular velocity w. Both inputs are supposed to be noisy, with gaussian noise $\epsilon \sim N(0, \sum)$ accounting for wheel slippages.

$$u = \begin{bmatrix} u_v \\ u_s \\ u_w \end{bmatrix} = \begin{bmatrix} v_\delta t \\ 0 \\ w\delta t \end{bmatrix} + \epsilon \quad \in se(2) \tag{4.10}$$

$$\sum = \begin{bmatrix} \sigma_v^2 \delta t & 0 & 0 \\ 0 & \sigma_s^2 \delta t & 0 \\ 0 & 0 & \sigma_w^2 \delta t \end{bmatrix} \quad \in \mathbb{R}^{33} \tag{4.11}$$

The robot pose is updated using the control input:

$$X_j = X_i \oplus u_j \tag{4.12}$$

We suppose that there are 3 landmarks noisy measurements that the robot can estimate at the same time with gaussian noise $n \sim N(0, N)$

$$y_k = X^{-1}.b_k + n \tag{4.13}$$

$$N = \begin{bmatrix} \sigma_x^2\delta & 0 \\ 0 & \sigma_y^2\delta \end{bmatrix} \quad \in \mathbb{R}^{22} \tag{4.14}$$

the landmarks positions are being calculated with reference to the robot frame and hence the multiplication with X^{-1} The landmarks position in the global frame are as follows

$$b_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \quad b_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad b_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix} \tag{4.15}$$

We use the LG-EKF prediction at each time step for the 3 landmarks given above :

$$\begin{aligned} \hat{X}_j &= \hat{X}_i \oplus u_j \\ P_j &= F P_i F^T + G \sum_j G^T \end{aligned} \tag{4.16}$$

with

$$\begin{aligned} F &= Ad_{Exp(u_j)^{-1}} \\ G &= J_r(u_j) \end{aligned} \tag{4.17}$$

and then we apply the algorithm given in the previous section :

$$\begin{cases} z & = y_k - \hat{X}^{-}1.b_k & ...innovation \\ Z & = HPH^T + N & ...Covarianceinnovation \\ K & = PH^T Z^{-1} & ...KalmanGain \\ \delta x & = Kz & ...Observederror \\ \hat{X} & \leftarrow \hat{X} \oplus \delta x & ...stateupdate \\ P & \leftarrow P - KZK^T & ...Cov.update \end{cases} \tag{4.18}$$

with

$$H = -[I \quad R^T[1]_x(b_k - t)] \tag{4.19}$$

the equations given above are very similar to the common kalman filter. However, It should be noticed that The matrices F, G,H are not the same and they are calculated with help of lie group theory

4.5.2 Results :

The initial conditions are as follows :

$$\hat{X}_0 = \begin{bmatrix} \hat{x} = 5 & \hat{y} = 2 & \hat{\theta} = \pi/3 \end{bmatrix} \quad X_0 = \begin{bmatrix} x_0 = 0 & y_0 = 0 & \theta_0 = 0 \end{bmatrix} \tag{4.20}$$

also we suppose there is a constant control input that controls the 2D robot:

$$u = \begin{bmatrix} 0.1 & 0 & 0.05 \end{bmatrix} \tag{4.21}$$

Here we show the position and the orientation of the robot along with the unfiltered position and orientation :

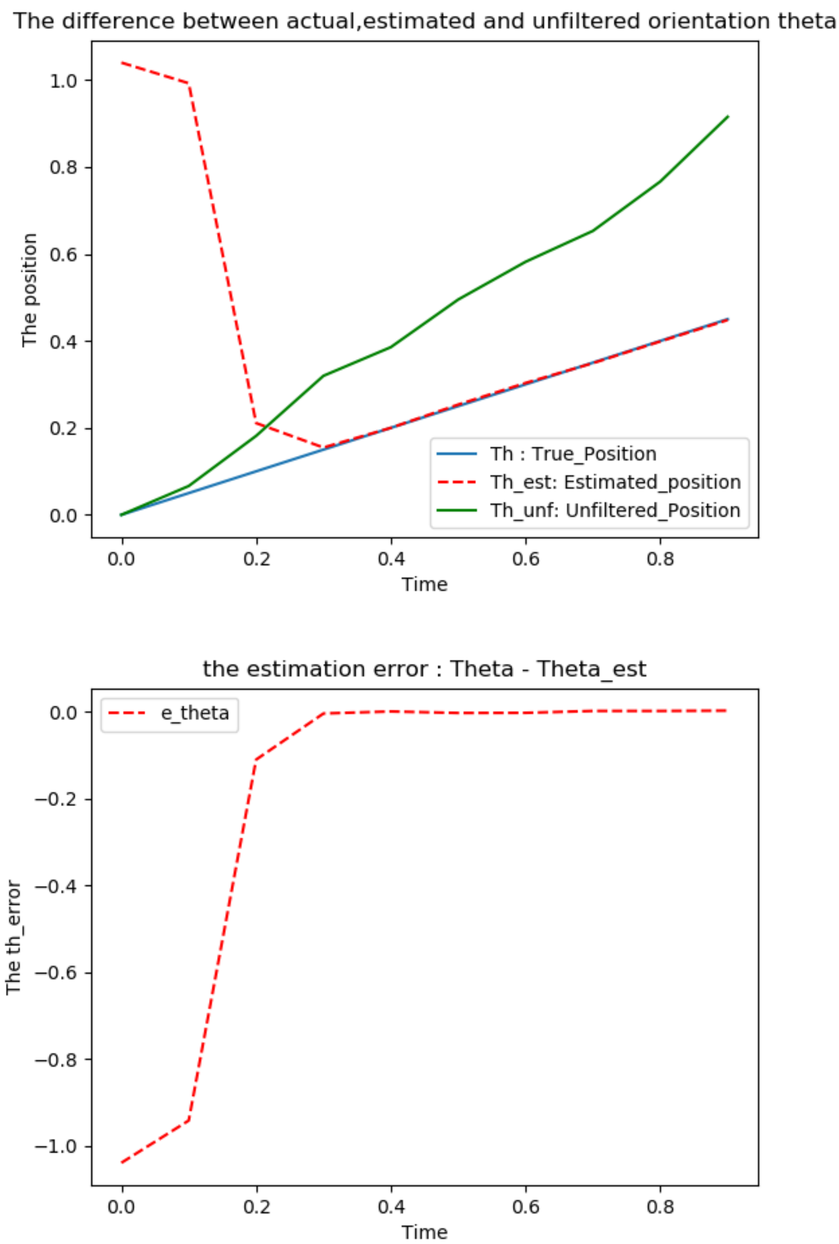


Figure 4.2: θ and e_θ

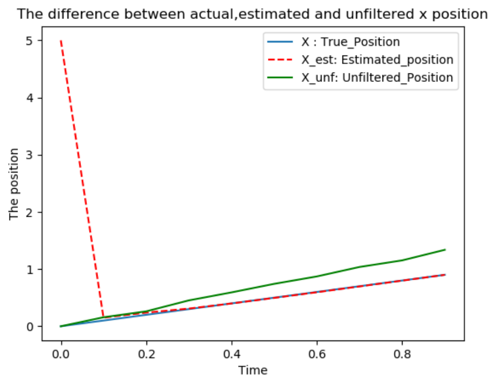


Figure 4.3: X position

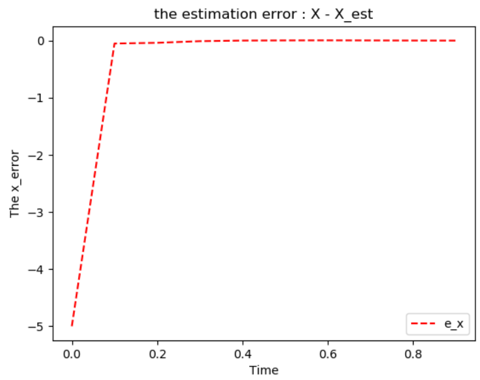


Figure 4.4: X position error

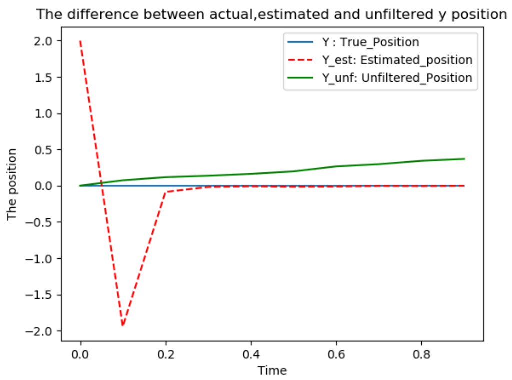


Figure 4.5: Y position

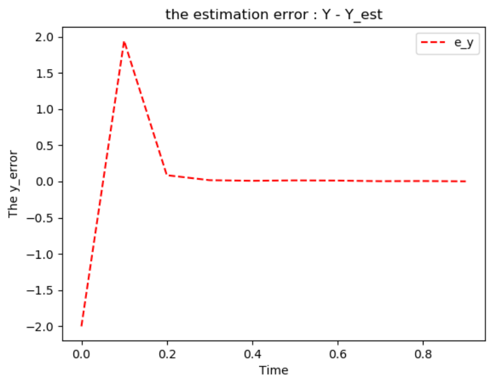


Figure 4.6: Y position error

4.5.3 Discussion and Conclusion :

We can see from the three previous graphs, that the Lie group Extended kalman filter follows the true position in a finite time despite the inconveniences that the system evolves on a manifold and the measurements are noisy as shown in green. The estimation converges for the three degrees of freedom, the X position, the Y position and the angle θ

4.6 Particle Filter :

4.6.1 Introduction :

Particle filter was popularized in the early 1990s, and has been utilized for the purpose, and its superiority comes from the fact that it can handle non linear and non-Gaussian systems which justifies its widespread usage in many fields such as global positioning of robots, self driving cars and so much more.The particle filtering technique refers to the procedure of finding a collection of random samples propagating in the state space to approximate the probability density function and substituting the integral operation with the sample mean to achieve the state lowest variance distribution.
the particles here can approximate any form of probability density distribution when

the number of particles is big enough. Intuitively, Particle filter, as its name indicates, constructs a lot of particles representing our guesses of a car's location and try to modify its guess as well as it can take into account the sequence of measurements obtainted. Moreover, particle filter methods are very flexible, easy to implement, parallelizable and applicable in very general settings.

4.6.2 Steps of the Particle Filter :

There are four essential steps in order to estimate the state $x \in \mathbb{R}^n$ using the particle filter

1. Generating a set of particles from a certain distribution

2. Spread the particles with the dynamics of the system

3. Update the probability of choosing a particle according to its closessness to the measurements

4. Resample the particles based on the new probability distribution

we begin explaining the first step :

1. Sampling :
   The first step is to sample from an initial probability distribution which is generaly gaussian , a set of N particles are initialized by a mean position or a mean state and specified deviation. The number N is chosen as a trade off between accuracy and speed of execution.
   it is presented here in this figure :

Figure 4.7: sampling

2. Prediction :
   Each particle represents a potential state of our system, using a model of the system, we can propagate the state of the system, which predicts the future states. The result of this is that the particles have moved to new locations and have spread out.

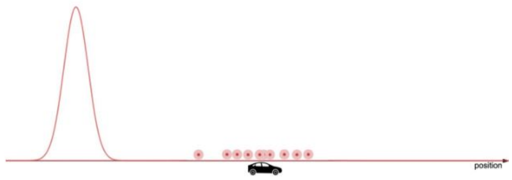


Figure 4.8: Prediction

3. Weight Update :
   We update the probability of picking up those particles using the difference between the particle and the observation and according to the closeness of the measurement

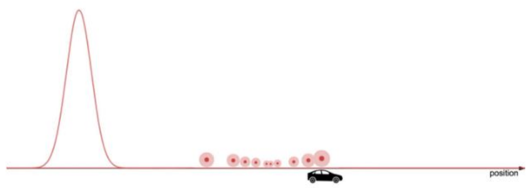


Figure 4.9: Weight Update

4. Resampling :
   If we pick a random element form the commulative probability distirbution which results in choosing the particles which have a higher weight more often than those that don't, this method is called resampling wheel which uses uniform distribution

Figure 4.10: Resampling

4.7 Algorithm and Simulation :

Algorithm 8 Particle Filter Algorithm :
Initialization : $X_0 \sim P(X_0)$
for i = 1, ... , N : sample $X_i \sim P(X_t, X_{t-1}^{(i)})$
endfor
for i = 1, ... , N :
evaluate importance weight $w_i = (Y_t, \tilde{X}_t)$
endfor
normalization of the importance weight for it to constitute a probability distribution
Resample :
for i = 1, ... , N :
Draw N with probability proportional to w_t

4.7.1 Simulation :

The particle filter has been simulated and tested on the following example :

$$\begin{cases} x = \frac{1}{2}x + \frac{25x}{1+x^2} + 8cos(1.2(t-1)) + V_n \\ z = \frac{x^2}{20} + W_R \end{cases}$$

using the same model

$$\begin{cases} x = \frac{1}{2}x + \frac{25x}{1+x^2} + 8cos(1.2(t-1)) \\ z = \frac{x^2}{20} \end{cases}$$

Using $N = 10$ samples we get the following performance:

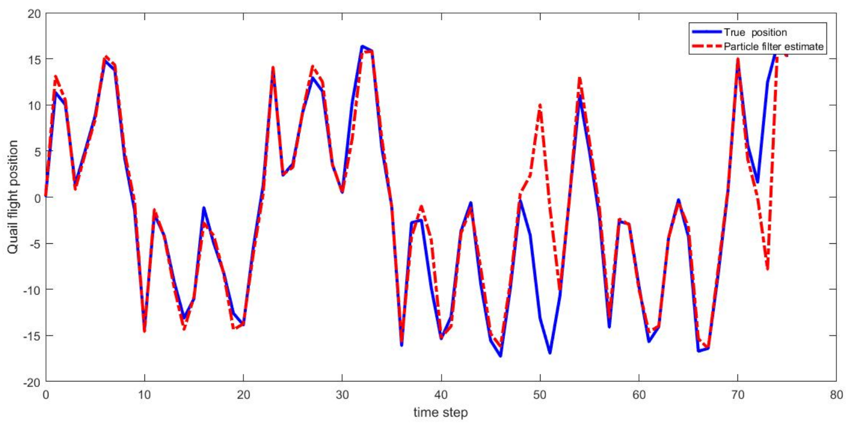


Figure 4.11: estimated using Particle filter N=10

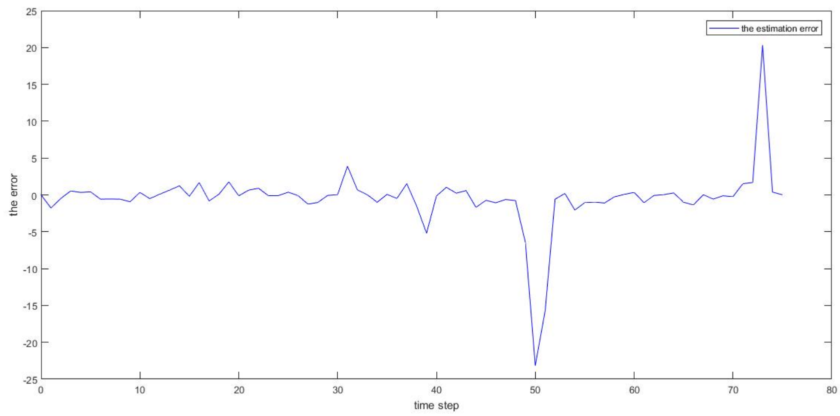


Figure 4.12: estimated using Particle filter N = 10

We will change the number of particle to see the advantages given by increasing the number of particles .

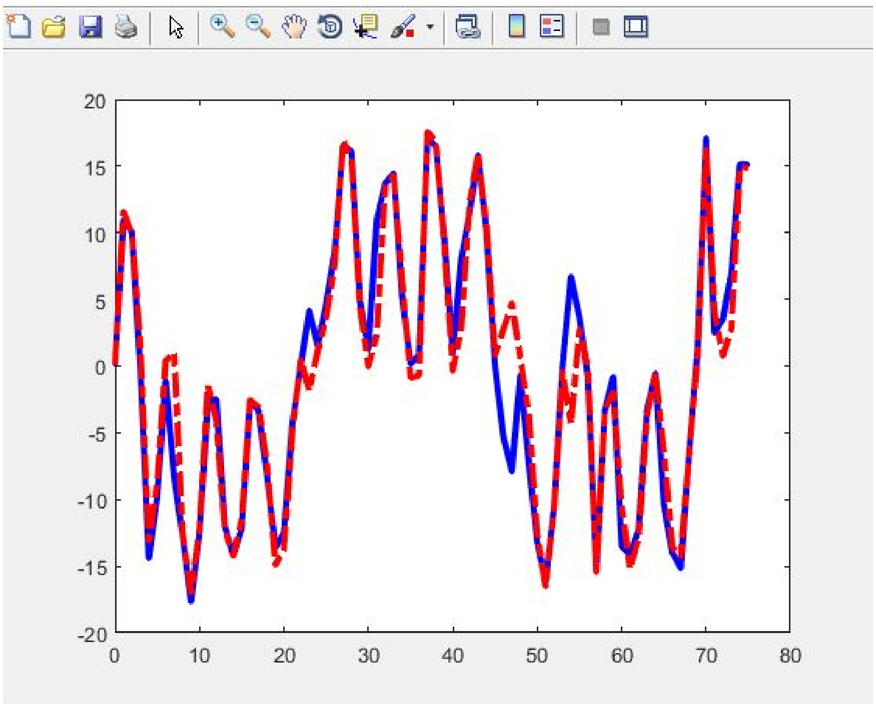


Figure 4.13: estimated using Particle filter N = 500

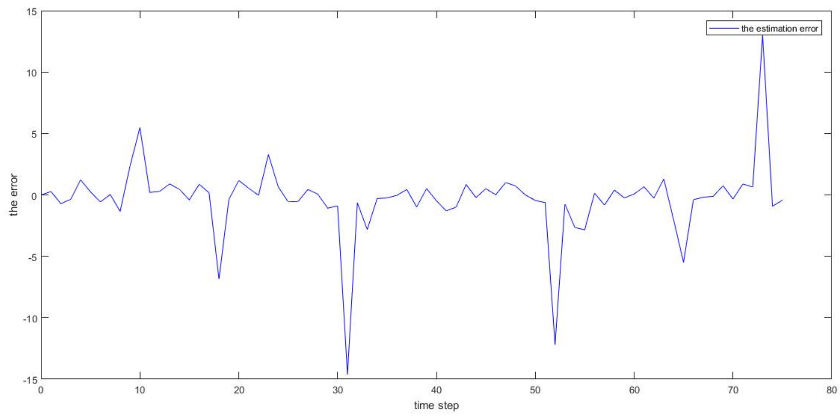


Figure 4.14: estimated using Particle filter N=500

4.7.2 Result discussion and constraints :

We can see that it behaves generally in the same way. However, the estimation is wrong at some points, this problem can be solved by increasing the number of particles used during the estimation.

By increasing the number of particles, the filter performance improves, by decreasing the error value from 25 when the number of particles was 10 to 15 when the number of particles increased to 500, this shows the efficiency of increased number of particles. The inconvenience to improving the performance by increasing the number of particles is that the computational cost also increases which limits its applicability for low cost microchips .

4.8 Particle Filter on Matrix Lie group :

The particle filter cited above can only be used for systems which evolve on $\mathbb{R}^n$, because the update equations tend to quickly leave the manifold, these equations do no hold and an extension of particle filter to systems evolving on lie groups is necessary.
Since we don't know the dynamics of the system, we can model the system mechanics evolving on a lie group as a stochastic differential equations (SDE).
Stochastic differential equations of the system evolving on a manifold :
First we must note that an ODE is written in the following form :

$$\frac{dx}{dt} = a(t)x(t)$$

When the function $a(t)$ is not known, we consider that the function have a noisy component $\epsilon(t)$ and so $a(t) = f(t) + \epsilon(t)$ where $f(t)$ is known, which gives rise to the following equation of Stochastic differential equation :

$$Dx(t) = f(t)x(t)dt + x(t)dW(t)$$

We suppose this system as a discrete system and we denote the state X_k as the state of the system at time step t_k, $X(t_k) = X_k$, and a measurement noise $Y_k = C(X_k) + E$ where

E is a noise on $\mathbb{R}^{nXl}$.
The objective of the particle filter is that it estimates the best possible candidate $\hat{X(t)}$ to the actual state $X(t)$ that is done by choosing the estimate which minimizes the next criteria :

$$E(d(X, X_N)) = \int d(X, X_k)^2 p(X_k|Y_{1:k}) d_G X_k$$

where d is the distance between two elements in the lie group, minimizing this integral which is a Mean Squared Error criteria, leads to finding the best possible element and hence the goal of the estimation is verified. However, calculating this integral and minimizing it is not simple, and so we use the tool of Monte Carlo methods to simplify this problem by sampling a set of N samples and using the theorem of large numbers that if N is big enough then we can approximate the expected value as follows:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N} X_i$$

$$E(d(X, X_N)) = \frac{1}{N}\sum_{i=1}^{N} d(X, X_N)$$

Another thing to take into account is that the samples that we have to take from must be from the $p(X_k|Y_{1:k})$ distribution, thus, a recursive procedure is needed which takes the samples from $P(X_{k-1}|Y_{1:k-1})$, so first we have to obtain the relationship between these two quantities which is given with the help of [27] by :

4.8.1 Particle filtering :

Let $S_k^1 ... S_k^N$ denote the N samples drawn from the $p(X_{k-1}|Y_{1:k-1})$ how to sample from $p(x_k|Y_{1:k-1})$ such that the samples remain on the manifold is discussed earlier
The next step is to normalize this probability distribution,

$$p_{k,s} = \frac{p(y_k|X_k = S_k)}{\sum_{i=1} N p(Y_k|X_k = S_k)}$$

The resampling algorithm is done by constructing the cumulative distribution P_N (a staircase function with steps at each s with height $p_{k,s}$), choosing a uniformly random point between
The state estimator $\hat{X_k}$

$$\hat{X_k} = argmin_{X \in G}(E(d(X, X_k)))$$

$$argmin_{X \in G}\frac{1}{N}\sum_{i=1}^{N} d(X, S_k)^2$$

Sampling $p(x_k|X_{k-1})$

$$X(t) = X_{k-1}e^{\Omega(t)}\Omega(t) \in g$$

Algorithm 9 PF-LG Algorithm :

Initialization : $X_0 \sim P(X_0)$

for i = 1, ... , N : sample $X_i \sim P(X_t, X_{t-1}^{(i)})$

endfor

for i = 1, ... , N :

evaluate importance weight $w_i = (Y_t, \tilde{X}_t)$

endfor

normalization of the importance weight for it to constitute a probability distribution

Resample :

for i = 1, ... , N :

Draw N with probability proportional to w_t

Part II

Part 2: Application on AUV

Chapter 5

Application : Modeling and Control of an AUV

5.1 Underwater vehicles and autonomy :

A UUV/AUV is an unmanned/autonomous underwater vehicle, which is a marine robot which has the capability of executing autonomous tasks without assistance. The need for such marine robots is that they help in many fields such as oceanography, military searching for downed airplanes, laying undersea cables. In terms of shape, the AUV can have a torpedo-like shape or a glider shape, or even bio-inspired AUVs in some cases. AUVs are given a certain mission to perform which require energy and power and High energy and high power source make missions with longer duration possible. However, these missions often reveal more problems because of the environmental changes that occur, such as the current waves, wheather conditions, and topography variation, the appearance of obstacles, and in some cases the UUV itself changes: such as instrument parameters, losing signal with a collaborative robot and so forth. All these problems show the necessity of more efficient and robust autonomous vehicles particularly to perform successful intelligent behaviours and adapt to unknown circumstances and respond to dynamic environments and achieve the assigned task whenever possible. This chapter is done with the help of the following articles: [30], [31],[32],[33],[34],[35],[36], [37],it has focused more throughtly on [38],[25],[39],[40],[41] The simulation is done with the help of the manifpy library constructed by Joan Sola

5.2 Types of underwater vehicle :

5.2.1 ROV :

ROV refers to Remotely Operated vehicles, which are remotely controlled vehicles, they are linked to a host ship and the communication is done by a neutrally buoyant long tether cables or, often when working in rough conditions or in deeper water, a load-carrying umbilical cable is used along with a tether management system (TMS), that transmits real-time video observations and environmental readings (e.g., depth, compass heading), they are used in survey and are devided into two categories, Class 1 for Observation Only and, Class 2: for observation with payload, but also in military use for ROVs have been used by several navies for decades, primarily for minehunting and minebreaking. Furthermore, the scientific community makes considerable use of ROVs to investigate the ocean, a number of deep sea animals and plants have been discovered or studied in their natural environment through the use of ROV. ROVs also are used for intervention by possessing manipulator arms adapted to the application for which ROV is destined to achieve.

Figure 5.1: ROV

5.2.2 HROV

HROV refers to Hybrid Remotely operated vehicles which are vehicles that can be autonomous due to integrated batteries but also have the caracteristics of a ROV

5.2.3 Bio-inspired vehicles

The Bio-inspired vehicles are vehicles in which their design and motion mechanism is influenced by animals and biology such as turtles, These types of vehicles are less common in the industrial world and they are found more in laboratories though they display a great efficiency in power consumption.

Figure 5.2: U-CAT

5.3 Modeling :

In reality, there are several techniques to modeling underwater vehicles, full-scale experiments, scaled experiments, empirical formula approximations and computational approaches.The robust control design depends on the mathematical description of underwater vehicle dynamics. To analyze the dynamic and hydrodynamic behaviour of UUVs, a model is required, meaning the interactive physics between the underwater vehicle and fluid.

Modeling of underwater vehicles involves two parts of study: kinematics and dynamics. The kinematics part describes the motion of the body without the forces and torques acted upon it but only describing the geometrical property of the system. In contrast, dynamics consider the torques and the forces acted on the body, there are many modeling techniques and articles but we will use those of [38]

Position, velocity v , force and torque vectors are expressed as follows :

$$\eta = [x, y, z, , \theta, \phi]^T$$

$$v = [u, v, w, p, q, r]^T$$

$$\tau = [X, Y, Z, K, M, N]^T$$

$\eta \in \mathbb{R}^6$ which represent both position and orientation, velocity $v \in \mathbb{R}^6$ represent both linear and angular velocity, $\tau \in \mathbb{R}^6$ represent the force and the torques.

Figure 5.3: Caption

5.3.1 Kinematics of AUVs :

The fundemantal branch of mechanics known as kinematics treats geometrical aspects of underwater vehicles. Which usually happens in 6 degrees of freedom, and the 6 different motion components are conveniently defined as: surge, sway, heave,roll, pitch and yaw. The NED frame and body-fixed frame are defined in the convential way as shown in figure. It is known that kinematic relation of velocity vector and position vector is expressed as the vectorial equation :

$$v = J(\theta)\dot{\eta} \tag{5.1}$$

Where $J(\theta)$ is the transformation between the Body fixed frame and the inertial frame which relates the times derivative of underwater vehicle position and angle to the translational and rotational velocities.

$$J(\theta) = \begin{bmatrix} R(\theta) & 0_{33} \\ 0_{33} & T(\theta) \end{bmatrix} \tag{5.2}$$

where $R(\theta) \in \mathbb{R}^{33}$ is the linear velocity transformation matrix, and $T(\theta) \in \mathbb{R}^{33}$ is the angular velocity transformation matrix.
From [38], we can see that :

$$T(\theta) = \begin{bmatrix} 1 & st\theta & ct\theta \\ 0 & c\phi & s\phi \\ 0 & \frac{s\phi}{c\theta} & \frac{c\phi}{c\theta} \end{bmatrix} \tag{5.3}$$

$$\mathrm{R}(\theta) = \begin{bmatrix} cc\theta & -sc\phi + cs\theta s\phi & ss\phi + cs\phi s\theta \\ sc\theta & cc\phi + s\phi s\theta s & -cs\phi + s\theta sc\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} (5.4)$$

Here we note that $T(\theta)$ is not defined at the pitch angle of $\frac{\pi}{2}$ which is called the singularity problem of euler angle representation. An alternate way is to use rotation matrices representation. a figure from [42] shows a more comprehensive illustration :

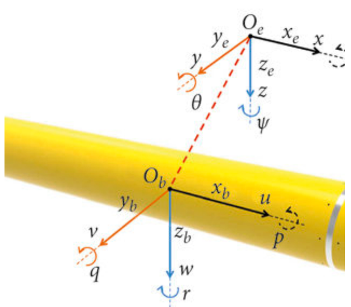


Figure 5.4: AUV kinematics

5.4 Rigid body dynamic of underwater vehicles:

Without the hydrodynamics forces which are caused by the water flowing against and around it. The underwater vehicle can be considered as a 6-DOF rigid body which obeys the Newton-Euler Equations. It is shown that the 6-DOFs underwater vehicle motion can be expressed as vectorial form as in the following :

$$M_{RB}\dot{b} + C_{RB}(v)v = \tau_{env} + \tau_p ro \tag{5.5}$$

The above vectorial forces and torques are considered to act on the vehicle's center of gravity, and they are balanced, Rigid-body mass inertia Matrix $M_{RB} \in \mathbb{R}^{66}$ is symmetric and positive definite, I_{33} is the identity matrix, and it is equal to the following :

$$M_{RB} = M_{RB}^T = \begin{bmatrix} mI_{33} & -mS(r_g^b) \\ mS(r_g^b) & I_0 \end{bmatrix}$$

$$M_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & I_x & -I_{xy} & -I_{xz} \\ mz_G & 0 & -mx_G & -I_{yz} & I_y & -I_{yz} \\ -my_G & mx_G & 0 & -I_{zx} & -I_{zy} & I_z \end{bmatrix}$$

I_0 is the inertia matrix and it is defined as follows:

$$I_0 = \begin{bmatrix} I_x & -I_{xy} & -I_{xz} \\ -I_{yz} & I_y & -I_{yz} \\ -I_{zx} & -I_{zy} & I_z \end{bmatrix}$$

defining M_{RB} as the following helps us find a definition of the C_{RB} matrix :

$$M_{RB} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

which makes the corolios matrix as the following :

$$C(v) = \begin{bmatrix} 0_{33} & -S(M_{11}v_1 + M_{12}v_2) \\ -S(M_{11}v_1 + M_{12}v_2) & -S(M_{21}v_1 + M_{22}v_2) \end{bmatrix}$$

with $v_1 = [u, v, w]^T$, $v_2 = [p, q, r]^T$

5.5 Hydrodynamics of underwater vehicles :

Hydrodynamics are important in the control design of underwater vehicles because of the complexity of the ocean environment. The hydrodynamic factors really make control design more challenging, which distinguishes underwater control design from air and land robots, and the types of hydrodynamic forces and torques which we can encounter in the ocean are as follows :

- Radiation Induced forces
- External disturbances: currents and waves
- Thruster propulsions

here we will add the hydrodynamic force to the principle equation added above :

$$M_{RB}\dot{b} + C_{RB}(v)v = \tau_{env} + \tau_{hydro}\tau_p ro \tag{5.6}$$

the total acting hydrodynamic of the underwater vehicle can be expressed according to [] as follows:

$$\tau_{hydro} = -M_A\dot{v} - C_A(v)v - D(|v|)v - g(\eta) \tag{5.7}$$

5.6 Gravitational and buoyant forces :

Generally a floating equalibrium is achieved by W which is the weight force and B which is the buoyancy force being equal on the same vertical line, designers of underwater vehicles take this into account, and so it is standard practice to design vehicles to be neutrally buoyant, which keeps the vehicle from sinking, and so their goal is also to balance between the center of gravity which is the point in which the gravity acts on, and the center of buoyancy, These two forces are here denoted as $g(\eta)$ and they are usually called restoring forces, because these forces are designed such as when an external force is applied to the vehicle, $g(\eta)$ can provide the torques to return to the equilibrium state, acting like a spring. $g(\eta)$ is as follows :

$$g(\eta) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -BG_yWcos(\theta)cos(\phi) + BG_zWcos(\theta)sin(\phi) \\ -BG_zWsin(\theta) + BG_zWcos(\theta)sin(\phi) \\ -BG_zWcos(\theta)sin(\phi) - BG_yWsin(\theta) \end{bmatrix} \tag{5.8}$$

5.7 Added Mass :

Added mass forces and moments, i.e., e induced by the surrounding fluid inertia.Generally, the added mass matrix $M_A \in \mathbb{R}^{66}$ is positive definite (fully sub- merged), and the diagonal elements of the matrix are positive. And experience has shown that $M_A = M_A^T$ is actually a good approximation [], For most practical applications, the off-diagonal elements of M_A will be small compared to the diagonal elements. Therefore, in most of the underwater vehicles, we can use the diagonal form of the added mass matrix M_A

$$M_A^{full} = -\begin{bmatrix} X_{\dot{u}} & X_{\dot{v}} & X_{\dot{w}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{v}} & Y_{\dot{w}} & Y_{\dot{p}} & Y_{\dot{q}} & Y_{\dot{r}} \\ Z_{\dot{u}} & Z_{\dot{v}} & Z_{\dot{w}} & Z_{\dot{p}} & Z_{\dot{q}} & Z_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{v}} & K_{\dot{w}} & K_{\dot{p}} & K_{\dot{q}} & K_{\dot{r}} \\ M_{\dot{u}} & M_{\dot{v}} & M_{\dot{w}} & M_{\dot{p}} & M_{\dot{q}} & M_{\dot{r}} \\ N_{\dot{u}} & N_{\dot{v}} & N_{\dot{w}} & N_{\dot{p}} & N_{\dot{q}} & N_{\dot{r}} \end{bmatrix} \tag{5.9}$$

and so :

$$M_A = -diag(X_{\dot{u}}, Y_{\dot{u}}, Z_{\dot{u}}, K_{\dot{u}}, M_{\dot{u}}, N_{\dot{u}})$$

It is common to separate the added mass forces and mo- ments in terms which belong to an added mass matrix .However, the coriolis matrix M_A of the added mass is neglected in front of C_{RB} and it is only mentionned for high speed underwater vehicles.

5.8 Damping :

Hydrodynamic damping matrix, D(|v|) $\in \mathbb{R}^{6X6}$, should be carefully involved in the underwater vehicle model, which is a function of linear and angular velocities, the effects of damping on complex shaped underwater vehicles is difficult to be accurately modeled. In modeling damping for AUVs as in [38], damping generally consists of 4 parts : Po- tential damping $D_P(v) \in \mathbb{R}^{3X3}$, skin friction$D_s(v) \in \mathbb{R}^{3X3}$, wave drift damping $D_w(v) \in \mathbb{R}^{3X3}$,nd vortex shedding damping $D_m(v) \in \mathbb{R}^{3X3}$, ,According to [114], if the underwater vehicle's velocities are suf- ficiently high $D(v)$ can be neglected. Assume the damping elements is not coupled, i.e., off-diagonal elements are negligible, then, the damping matrix $D_P(|v|) \in \mathbb{R}^{3X3}$,can be simplified into a diagonal matrix:

$$M_A = -diag(X_{u|u|}|u|, Y_{v|v|}|v|, Z_{w|w|}|w|, K_{p|p|}|p|, M_{q|q|}|q|, N_{r|r|}|r|)$$

5.8.1 Environmental Disturbances :

Actually, the wind, waves and current of the ocean are really complex, including additive and multiplicative types of random disturbances. However, in practice a good assumption according to [38] are, the wind, wave and currents are considered to be linearly superposed for marine vehicles, which separates the effects into linear components.

5.9 6 DOFs Underwater vehicle Model [38] :

The dynamic and hydrodynamic underwater vehicle model in the principal equation are equations which are expressed in the body-fixed frame, here we express the previous model in the earth-fixed frame based on the inverse jacobian as follows :

$$M^*\ddot{\eta} + D^*(|v|)(\dot{\eta}) + g^*(\eta) = \tau^*_{pro} + \tau^*_{env} \tag{5.10}$$

where :

$$M = M_{RB} + M_A$$

$$M^* = J^{-T}(\theta)MJ^{-1}(\theta)$$

$$D^*(|v|) = J^{-T}(\theta)D(|v|)J^{-1}(\theta)$$

$$g^*(\eta) = J^{-T}g(\eta)$$

$$\tau^*_{pro} = J^{-T}\tau^*_{pro}$$

$$\tau^*_{env} = J^{-T}\tau^*_{env}$$

these equations will later be used to do a numerical simulation of an AUV on a Matlab

5.10 Discussion :

In this chapter, Based on the Fossen marine vehicle formulas, a mathematical description of the dynamic and hydrodynamic underwater vehicle model is provided.The discussion of kinematics, or the principles for transforming between body-fix reference and inertial reference, comes first.Then, on the assumption that all components may be linearly superposed, the kinetics with classical rigid-body dynamic and hydrodynamic components is described Finally, environmental disturbance are provided. Finally, the inertial reference presents the six degrees of freedom underwater vehicle model.

5.11 Control of an autonomous underwater vehicle :

In recent years, the majority of research has been devoted to the control strategy of autonomous underwater vehicles which is very important, different control strategies are : path following, way point tracking, trajectory and localization. The control strategy is chosen based on the given mission, and very often instead of single AUV multiple AUVs are employed to achieve higher efficieny and more complex missions.
In order to achieve the path following control of an AUV, the error between the path parameters and AUV position and orientation should be reduced to zero.
This is a difficult problem since the complete dynamics is a nonlinear 6DOF equation of motion with coupled and nonlinear terms which are generally hard to model accurately depending on the shape and the structure of the AUV itself as discussed earlier,

This makes it hard for linear controllers to provide an efficient performance, and although some investigations employing the [27] feedback linearization method for path following control but they are only suitable for some operating points. Another problem occurs, when we consider the underactuated case which is even harder than the problem of controlling fully actuated systems, meaning that the number of control inputs are less than the degrees of freedom, and due to external disturbances, such as currents and waves, it is difficult to achieve the path following control strategies,
a control strategy using a nonlinear PID of a L2ROV using the standard modelisation of AUVs, and another control strategy using modelisation on lie groups is provided in this chapter

5.12 Non linear PID control of AUV

In this section, we will control a type of HROV using a non linear PID controller based on saturation function and varying parameters, this controller is based on set point regulation and the proof of the stability is given via lyapunov stability. The control scheme is validated using numerical simulation with matlab.

5.12.1 Control of a L2ROV :

The L2ROV is a remotely operated vehicle built at the university monpellier 2 and here we will discribe its dynamic model. To simplify our simulation of this vehicle we will assume that the vehicle is moving at low speeds, leading to a more simplified dynamics The L2ROV is a tethered underwater vehicle, whose size is about 75cm long, 55cm width, and 45cm height. The propulsion system of this underwater vehicle consists of six thrusters, there are two kinds of motion the translational motions which are surge,sway, and heave,

while the rotational motions are roll, pitch, and yaw
The L2ROV has 6 propellers which makes it a fully actuated underwater vehicle. The surge motion is generated by the sum of the forces created by T_4 and T_5 , sway movement is actu- ated by T_6 , and heave is produced by the sum of thrusts of T_1 , T_2 and T_3 . The roll movement is actuated through differential force of the thrusters T_2 and T_3 ; the pitch motion is obtained similarly using thrusters T_1, T_2 and T_3 , and the yaw motion is generated by T_4 and T_5 the technical properties which are used for the simulation via matlab are obtained with the help of [38] :

- Mass 28 kg
- Floatability 9N
- Maximal depth 100m
- Thrusters 6 Seabotix BTD150
- cont. bollard thrust = 2.2kgf each with Devantech MD03 drivers
- Power 48V - 600W
- Light 2 x 50W LED
- Attitude sensor Sparkfun Arduimu V3 Invensense MPU-6000 MEMS 3-axis gyro and accelerometer 3-axis I2C magnetometer HMC-5883L Atmega328 microprocessor
- Camera Pacific Corporation VPC-895A CCD1/3” PAL –25–fps
- Depth sensor Pressure Sensor Breakout-MS5803-14BA
- Sampling period 50ms
- Surface computer Dell Latitude E6230 - Intel Core i7 - 2.9GHz Windows 7 Professional 64 bits Microsoft Visual C++ 2010
- Tether length 150m

Figure 5.5: U-CAT

5.13 dynamic modeling :

the dynamics of the vehicle is obtained from the last chapter which has the next equation :

$$M^{*}\ddot{\eta} + D^{*}(|v|)(\dot{\eta}) + g^{*}(\eta) = \tau^{*}_{pro} + \tau^{*}_{env} \tag{5.11}$$

where $M \in \mathbb{R}^{66}$ is the inertia matrix ,$C(v) \in \mathbb{R}^{66}$ is the Coriolis-centripetal Matrix. Since we are assuming that the vehicle is moving at low speeds then this matrix is neglected, $D(v) \in \mathbb{R}^{66}$ is the damping matrix $g(\eta)$ describes the vector of restoring forces and moments, $J(\eta) \in \mathbb{R}^{66}$ is the transformation matrix mapping from the body-fixed-frame to earth-fixed-frame

5.14 Inertia and damping matrices :

$M = M_{RB} + M_{A}$ and

$$M = diagm(-X_{\dot{u}}, m - Y_{\dot{v}}, m - Z_{\dot{u}}, I_{xx} - K_{\dot{p}}, I_{yy} - M_{\dot{q}}, I_{zz} - N_{\dot{r}})$$

for this type of vehicle and from [38] we have obtained the following values for the inertia matrix :

$$I_0 = \begin{bmatrix} 0.35 & -0.02 & -0.04 \\ -0.02 & 0.69 & -0.02 \\ -0.04 & -0.02 & 0.65 \end{bmatrix}$$

we consider the following modelisation of damping for low-speed underwater vehicles :

$$D(v) = diag(x, y, z, k, m, n)$$

for this type of vehicle and according to the following article [38] we have that:

$$D(v) = diag(30, 70, 80, 1.4, 2.5, 2.9)$$

5.14.1 The restoring forces and moments :

The restoring forces and moments are generated by the weight and buoyancy force : $f_B = -[00B]^T$ and the weight force : $f_W = [00W]^T$ once these two forces are obtained, which are with respect to the earth fixed frame, using the transformation matrix $J_1(\eta_2) = R_{z,} R_{y,\theta} R_{x,\phi}$ it can be expressed with respect to the body-fixed frame :
$F_B = J_1(\eta_2)^{-1} f_B$ 15mm $F_W = J_1(\eta_2)^{-1} f_W$

thus the total force is :

$$f_g = \begin{bmatrix} (B-W)sin(\theta) \\ (W-B)cos(\theta)sin(\phi) \\ (W-B)cos(\theta)cos(\phi) \end{bmatrix} \tag{5.12}$$

on the other hand the total torque depends on the position of center of gravity and the center of buoyancy :

$$m_g = r_w F_w + r_b F_B$$

r_b and r_w represent the positions of CG and CB respectively. The design of L2ROV makes the buyoancy force greater than the weight force obtaining the equation below for the restoring forces :

$$g(\eta) = \begin{bmatrix} f_g \\ m_g \end{bmatrix} = \begin{bmatrix} f_b sin(\theta) \\ -f_b cos(\theta)sin(\phi) \\ -f_b cos(\theta)cos(\phi) \\ -z_b B cos(\theta)sin(\phi) \\ -z_b B sin(\theta) \\ 0 \end{bmatrix} \tag{5.13}$$

5.14.2 Total forces and torques of the propulsions :

Given the design of the L2ROV vehicle the forces that can be generated from the propulsions are as follows :

$$\tau_1 = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} f_4 + f_5 \\ f_6 \\ f_1 + f_2 + f_3 \end{bmatrix} \tag{5.14}$$

The torques however have different set of equations since they rely on the position the force is applied on.

$$\tau_2 = \sum_{i=1}^{6} l_{ii} \tag{5.15}$$

$$\tau_2 = \begin{bmatrix} \tau_k \\ \tau_M \\ \tau_N \end{bmatrix} = \begin{bmatrix} l_{2y} f_2 + l_{3y} f_3 \\ l_{2x} f_2 + l_{3x} f_3 + l_{1x} f_1 \\ l_{4y} f_4 + l_{5y} f_5 \end{bmatrix}$$

and the vector of control inputs is the following :

$$\tau = \begin{bmatrix} f_4 + f_5 \\ f_6 \\ f_1 + f_2 + f_3 \\ l_{2y} f_2 + l_{3y} f_3 \\ l_{2x} f_2 + l_{3x} f_3 + l_{1x} f_1 \\ l_{4y} f_4 + l_{5y} f_5 \end{bmatrix} \tag{5.16}$$

5.14.3 Non linear PD control of L2ROV :

The dynamics equations for the system as developped in the previous chapter is as follows:

$$M\dot{v} + C(v)v + D(v)v + g^*(\eta) = \tau^*_{pro} + \tau^*_{env} \tag{5.17}$$

$$\dot{\eta} = J(\eta)v \tag{5.18}$$

The non linear PD controller that we will be interested in is :

$$\tau = g(\eta) - J^T \tau_{PD} \tag{5.19}$$

$$\tau_{PD} = K_p e(t) + K_d \frac{de(t)}{dt} \tag{5.20}$$

here K_p and K_d are diagonal positive definite matrices, and $e(t) = \eta - \eta_d$

the saturation function is applied to the control law giving us a control law of the form :

$$\tau_{NLPD} = \sigma_{b_p} K_p e(t) + \sigma_{b_d} K_{\frac{de(t)}{dt}} \tag{5.21}$$

in here we will modify the saturation function as to become slightly more efficient in our case just like in [38]:

$$\sigma_b(h) = \begin{cases} \hat{b}, h > \hat{b} \\ h, |h| \leq \hat{b} \\ -\hat{b}, h < -\hat{b} \end{cases} \tag{5.22}$$

$$u_i = \begin{cases} \hat{b_i}, k_i h_i > \hat{b_i} \\ k_i h_i, k_i h_i > \hat{b_i} \\ -\hat{b_i}, k_i h_i < -\hat{b_i} \end{cases} \tag{5.23}$$

$$u_i = \begin{cases} sign(h_i)\hat{b_i} if |h_i| > d_i \\ \hat{b_i} d_i^{-1} h_i if |h_i| \leq d_i \end{cases} \tag{5.24}$$

developing even further gives the following

$$u_i = \begin{cases} b_i|h_i|^{-1}h_i if|h_i| > d_i \\ \hat{b}_i d_i^{-1}h_i if|h_i| \leq d_i \end{cases} \tag{5.25}$$

In Some cases, the control law can't bring the state to the desired one because of its boundedness property and that's why we introduce the following variation of the saturation function:

$$u_i = \begin{cases} b_i|h_i|^{-1}h_i if|h_i| > d_i \\ \hat{b}_i d_i^{-1}h_i if|h_i| \leq d_i \end{cases} \tag{5.26}$$

In short, the modified non linear PD controller is the following :

$$\tau_{NLPD} = k_{pf}(.)e_j(t) + k_{df}(.)\frac{de_j(t)}{dt} \tag{5.27}$$

where :

$$K_{pj}(.) = \begin{cases} b_{pj}|e_j(t)|^{\mu_p - 1} if|e_j(t)| > d_{pj} \\ b_{pj}d_{pj}^{\mu_p - 1} if|e_j(t)| \leq d_{pj} \end{cases} \tag{5.28}$$

and

$$K_{dj}(.) = \begin{cases} b_{dj}|e_j\dot{(t)}|^{\mu_d - 1} if|e_j(t)| > d_{dj} \\ b_{pj}d_{dj}^{\mu_d - 1} if|e_j(t)| \leq d_{dj} \end{cases} \tag{5.29}$$

Theorem :
the control law : $\tau = g(\eta) - J^T(\eta)(K_p(.)e + K_d(.)\dot{e})$ if k_p and k_d are defined as in the previous equations, the system is asymptotically stable
Proof :
for set-point regulation $\dot{\eta_d} = 0$ and $\dot{e} = \dot{\eta}$, knowing this the equation of the control law τbecomes:

$$\tau = g(\eta) - J^T(\eta)(K_p(.)e + K_d(.)\dot{\eta})$$

applying the control law to the dynamics equation gives:

$$M\dot{v} + C(v)v + D(v)v + g^*(\eta) = -J^T(\eta)(K_p(.)e + K_d(.)\dot{\eta})$$

$$M\dot{v} + C(v)v + D(v)v = -J^T(\eta)(K_p(.)e + K_d(.)J(\eta)v) \tag{5.30}$$

we define : $K_{dd}(.) = J^T(\eta)K_d(.)J(\eta)$ and hence the equation becomes :

$$M\dot{v} + C(v)v + D(v)v = -J^T(\eta)K_p(.)e - K_{dd}(.)v) \tag{5.31}$$

The closed loop system is then :

$$\frac{d}{dt}\begin{bmatrix} e \\ v \end{bmatrix} =$$

$$\begin{bmatrix} J(\eta)v \\ M^{-1}(-J^T(\eta)K_p(.)e - K_{dd}(.)v - C(v)v - D(v)v) \end{bmatrix}$$

the lyapunov candidate : from [38] we can propose the next lyapunov candidate :

$$V(e, v) = \frac{1}{2}v^T Mv + \int_0^e \epsilon^T K_p(\epsilon)d\epsilon$$

according to lemma 2 from [38] we have :

$$\int_0^e \epsilon^T K_p(\epsilon) d\epsilon > 0 \forall e \neq 0 \in \mathbb{R}^n$$

Therefore the lyapunov function is globally positive definite and unbounded

$$V(\dot{e}, v) = v^T M\dot{v} + e^T K_p(e) J(\eta) v$$

$$V(\dot{e}, v) = -v^T J^T(\eta) K_p(e) e - v^T K_{dd}(\eta, \dot{e}) v - v^T C(v) v - v^T D(v) v + e^T K_p(e) J(\eta) v$$

since K_p is symmetric and $C(v)$ is antisymmetric then :

$$V(\dot{e}, v) = -v^T (K_{dd}(\eta, \dot{e}) + D(v)) v$$

since $K_d > 0$ therefore $K_{dd} = K_{dd}^T > 0$ and since our assumption about D makes it $D(v) > 0$ then $V(\dot{e}, v)$ is a globally negative semidefinite.

finally, we use LaSalle theorem to conclude that the equilibrium point is asymptotically stable :

$$\Omega = \begin{bmatrix} e \\ v \end{bmatrix} : V(\dot{e}, v) = 0$$

introducing $v = 0$ and $\dot{v} = 0$ into the equation leads to the unique invariant point $e = 0$ completing the proof .

5.15 Simulation :

We have deleppoed a simulated equivalent of the L2ROV with Matlab, using the equations The simulation has reveled the following results : The desired state is :

$$X_d = \begin{bmatrix} 1 & 0 & 0 & pi/4 & 0 & 0 \end{bmatrix} \tag{5.32}$$

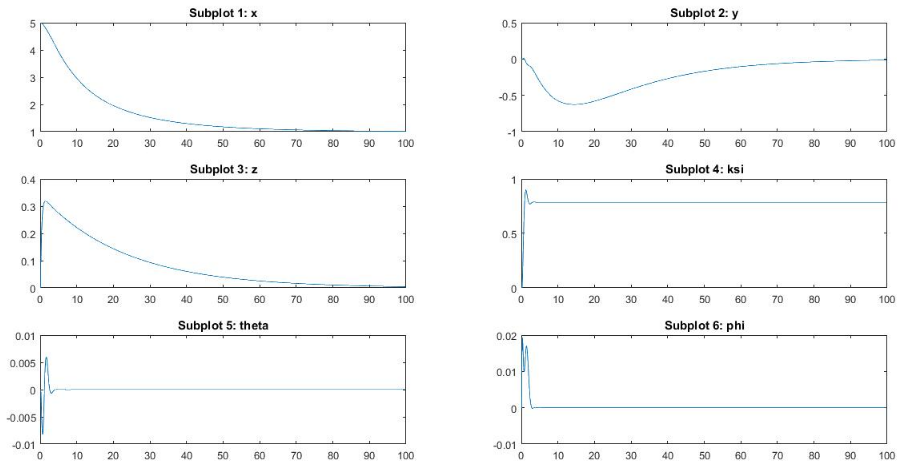


Figure 5.6: The state X

$$X_d = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.33}$$

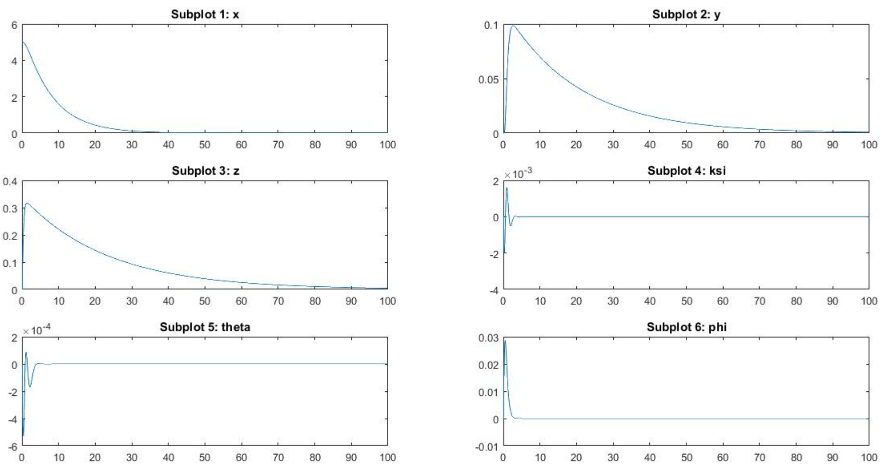


Figure 5.7: The state X for a different desired state

5.15.1 Discussion :

We see that the nonlinear PD controller is efficient and takes the Autonomous Underwater Vehicle to the desired state, a comparison between this and control based on lie group is difficult to accomplish since each of them evolve on different spaces. However, experimental results could show us which method is more efficient which is most likely the manifold representation .

5.15.2 LG-PD control on SE(2) based on EKF on Matrix Lie group of Autonomous Underwater Vehicle :

In the following section, we propose a controller design based on proportional derivative on Lie group based on an Lie group EKF observer. In here we treat the problem of controling the position and orientation of an AUV on the plane . First of all, we suppose that the AUV system evolves on the following lie group SE(2), such that the modeliaztion is goverened by the following equation : $\dot{X} = X\hat{w}$
Since the measurements are frequently not reliable, and for industrial applications that need inexpensive sensors, the estimation problem is obligatory. We treat the problem of estimation based on matrix Lie Group using the same procedure as the EKF on Lie Group discussed in the previous chapter, by using the measurements of the landmarks given by the system' sensors, we can estimate the AUV position and orientation, we utilize this information to control the AUV by the twist and linear velocity which are supposed in this case to be our control inputs,

5.15.3 Scheme :

The scheme of the simulation is given in the following figure :

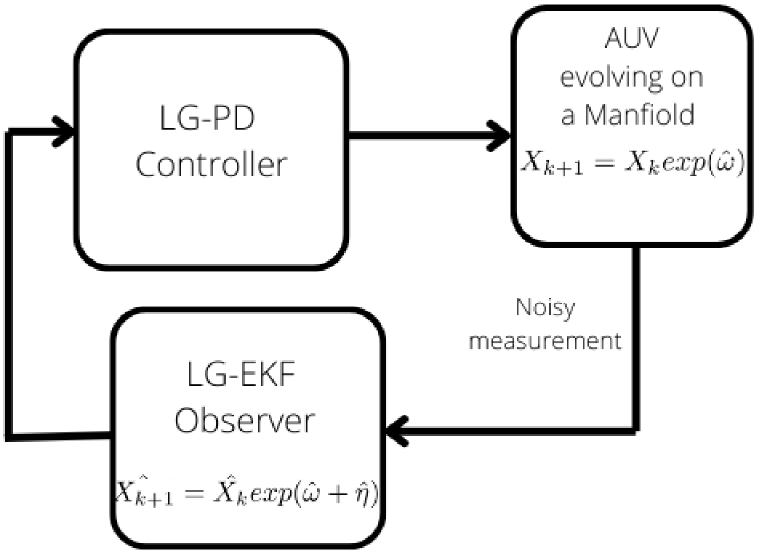
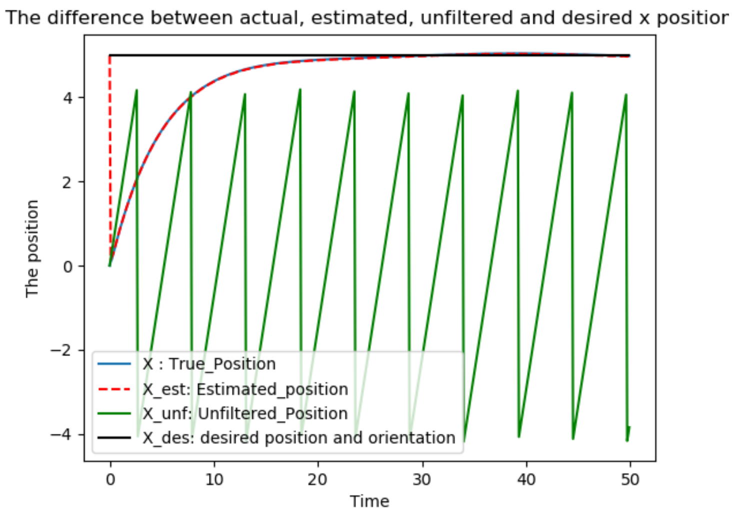
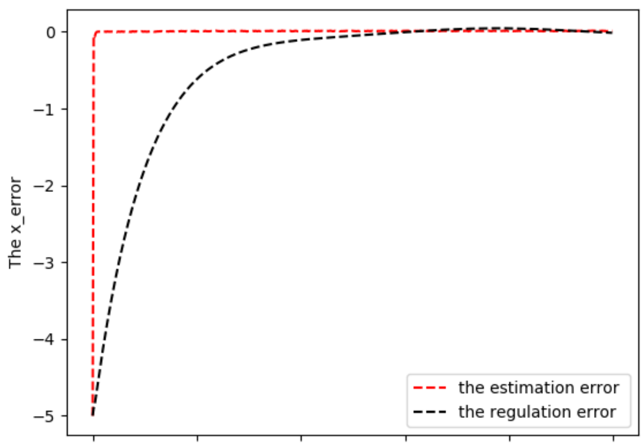


Figure 5.8: Observer based controller on lie groups

5.15.4 Results :





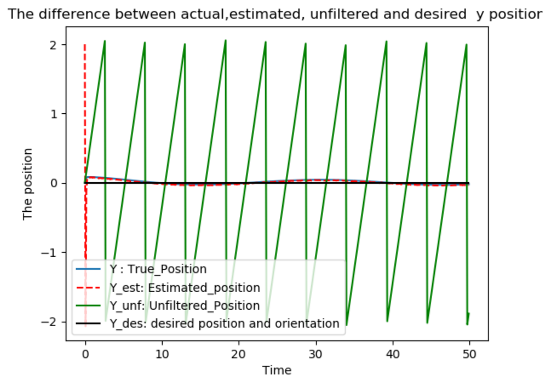


Figure 5.10: Y position

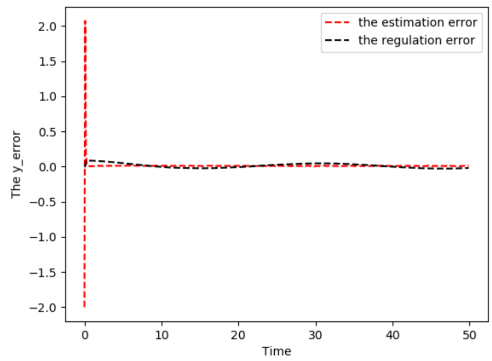


Figure 5.11: e_y

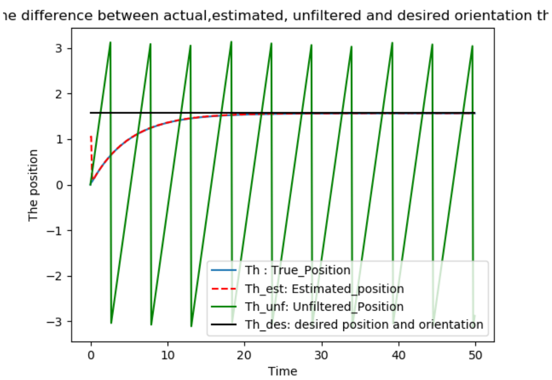


Figure 5.12: orientaion θ

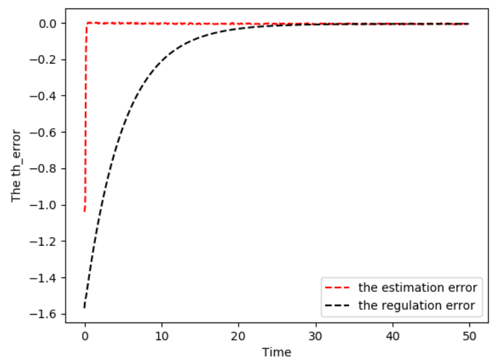


Figure 5.13: e_θ

5.15.5 Discussion :

The performance of the observer-based controller is well, by converging of the estimated state to the actual state and eliminating all the noisy measurements which are shown in green. From the regulation side, we see that the observer-based controller converges to the desired state in finite time with a dynamic of a first order system. this validates the performance of the proposed controller and shows the potential of the Lie Group theory application in practical situations .

5.16 Conclusion :

Throughout this chapter, we provided a classical modelization of autonomous underwater vehicle, along with regulation of its position and orientation with a nonlinear PD controller which brings the state to the desired position. Secondly, the dynamical system has been modeled on Matrix Lie group in SE(2) and another controller has been provided which is the observer-based controller based upon the extended kalman filter on matrix lie group using 3 landmarks, this controller also showed its performance compared to the other NLPD control validating the power of Lie groups on Control and estimation .

Chapter 6

General Conclusion :

In this work, we have discovered the potential of Lie Theory. The necessity of describing systems as objects evolving on lie groups have made the working with lie groups very interesting and pushed the robotics community to make a considerable effort to correctly describe estimate issues. Many advantages can be accomplished. For example, SO(3) representation of attitude is very useful since it removes singularity and uniqueness problems. To fully grasp the potential given by Lie group theory, one has to analyse different ascpects from it such as control and estimation.

In this project, we presented a EKF on Matrix Lie groups based PD controller for the autonomous underwater vehicle . We first provided a more thorough mathematical derivation of the concepts of Lie theory and Lie groups. We then introduced the control problem on lie groups by proposing three different control schemes : PD controller, Sliding mode controller, LQR controller. We then brought our attention to observers on Matrix lie group such as the extended kalman filter, we also provided simulation and discussion of such an observer.Finaly, we proposed an observer-based controller for the control of an AUV on SE(2).

```
dt = 0.1
N = 500
%
u = [0;0;0]   %% initializing the control vector

theta_des = pi/2
w = [0;0;theta_des]         %%% desired orientation (the length of the
    vector is the angle)
X_des = expm(hat(w))        %%% desired orientation matrix
X = eye(3)                  %%% Initial Value

kp = 0.1
zero_3d = [0;0;0]
thet =[]
erreur = []
U = []
for i=1:N
    u = -dt*kp*Lie_log(X_des'*X)
    U = [U u];
    X = expm(hat(u))*X                      %% updating the matrix
    inter = Lie_log(X)
    thet = [thet inter(3)]
end
t=0:dt:dt*(N-1);
plot(t,thet)
legend('theta')
figure(2)
plot(t,U,'--r')
legend('u')
```

```
dt = 0.1;
N = 2000;
%

u=[0;0;0];

%% desired attitude and angular velocity
theta_des = pi/3 ;
w = [0;0;theta_des]  ;      %%% desired orientation (the length of the
    vector is the angle)
R_d = expm(hat(w)) ;      %%% desired orientation matrix

w_d = [0;0;0];

%% initial conditions
X = eye(3) ;                %%% Initial Value
Wx = hat(w);                %%% deriving wx from w
R = X;

K = 50;
J = [3 0 0;0 4 0;0 0 5];

Theta=[];
W = [];
U = []

k_p = 0.2
```

```
k_d = 0.3

for i=1:N
    %% system dynamics
    Wx = hat(w);
    R = R*expm(dt*Wx);
    w = w+dt*inv(J)*(cross(J*w,w)+u)
    %% the attitude and velocity error
    R_e = R_d'*R
    w_e = w-R_e'*w_d

%     %% the control law based on the sliding mode controller

    u = -cross(J*w,w) + J*(-k_p*Lie_log(R_e) - k_d*w_e) ;    %% the control law
    U = [U u];
    W = [W w];
    Theta=[Theta Lie_log(R)];
end
%
t=0:dt:dt*(N-1);
figure(1)
plot(t,Theta,'b')
legend('Theta')
vect = zeros(3,N);
vect(3,:)=theta_des;
error = Theta - vect;
figure(2)
plot(t,error,'r')
legend('Error')
%
figure(3)
plot(t,U)
% legend('control effort')
```

```
Q= [10 0;0 1]; R=0.1; b=0.1;J=1.0    %% initializing Q,R of the system
theta_d = pi/2               %% desired theta
A=[0 1;0 -b/J]
B=[0;1/J]
K = [10 5.378]
%% initializing x
Phi = eye(2)
w= 0

Td = 0;wd=0    %% desired torque and angular velocity

N=500
dt=0.1

Theta = []
W = []
TORQUE = []
for i=1:N
    Phi_d=expm(hat(theta_d))
    x_tilt = [Lie_Log_so2(Phi*inv(Phi_d));w-wd]
    T_tilt = -K*x_tilt
    T = T_tilt + Td
```

```
    w_hat = hat(w)
    Phi = Phi*expm(w_hat*dt)
    w = w+dt*(T/J-b*w/J)

    if(i ==250)
        theta_d = -pi/2                  %% second desired theta
    end

    Theta = [ Theta Lie_Log_so2(Phi)]
    W = [W w];
    TORQUE = [TORQUE T];
end
t=0:dt:dt*(N-1)
figure(1)
plot(t,Theta)
legend('Theta')
figure(2)
plot(t,W,'r')
legend('angular velocity w')
figure(3)
plot(t,TORQUE,'g')
legend('Torque')
```

```



%% clear everything
clear all
close all
clc


%% initialize the variables
set(0,'DefaultFigureWindowStyle','docked') %dock the figures..just a
    personal preference you don't need this.
x = 0.1; % initial actual state
x_N = 1; % Noise covariance in the system (i.e. process noise in the state
    update, here, we'll use a gaussian.)
x_R = 1; % Noise covariance in the measurement (i.e. the Quail creates
    complex illusions in its trail!)
T = 75; % duration the chase (i.e. number of iterations).
N = 10000; % The number of particles the system generates. The larger this
    is, the better your approximation, but the more computation you need.



V = 2; %define the variance of the initial esimate
x_P = []; % define the vector of particles

% make the randomly generated particles from the initial prior gaussian
    distribution
for i = 1:N
    x_P(i) = x + sqrt(V) * randn;
end

%{
%show the distribution the particles around this initial value of x.
```

```
figure(1)
clf
subplot(121)
plot(1,x_P,'.k','markersize',5)
xlabel('time step')
ylabel('flight position')
subplot(122)
hist(x_P,100)
xlabel('flight position')
ylabel('count')
pause
%}



z_out = [x^2 / 20 + sqrt(x_R) * randn];  %the actual output vector for
    measurement values.
x_out = [x];  %the actual output vector for measurement values.
x_est = [x]; % time by time output of the particle filters estimate
x_est_out = [x_est]; % the vector of particle filter estimates.



for t = 1:T

    x = 0.5*x + 25*x/(1 + x^2) + 8*cos(1.2*(t-1)) +  sqrt(x_N)*randn;
    z = x^2/20 + sqrt(x_R)*randn;

    for i = 1:N


        z_update(i) = x_P_update(i)^2/20;

        P_w(i) = (1/sqrt(2*pi*x_R)) * exp(-(z - z_update(i))^2/(2*x_R));
    end


    P_w = P_w./sum(P_w);

    %{
    figure(1)
    clf
    subplot(121)
    plot(P_w,z_update,'.k','markersize',5)
    hold on
    plot(0,z,'.r','markersize',50)
    xlabel('weight magnitude')
    ylabel('observed values (z update)')
    subplot(122)
    plot(P_w,x_P_update,'.k','markersize',5)
    hold on
    plot(0,x,'.r','markersize',50)
    xlabel('weight magnitude')
    ylabel('updated particle positions (x P update)')
    pause
```

```
    %plot the before and after
    figure(1)
    clf
    subplot(131)
    plot(0,x_P_update,'.k','markersize',5)
    title('raw estimates')
    xlabel('fixed time point')
    ylabel('estimated particles for flight position')
    subplot(132)
    plot(P_w,x_P_update,'.k','markersize',5)
    hold on
    plot(0,x,'.r','markersize',40)
    xlabel('weight magnitude')
    title('weighted estimates')
    %}
    %% Resampling: From this new distribution, now we randomly sample from
   it to generate our new estimate particles

    for i = 1 : N
        x_P(i) = x_P_update(find(rand <= cumsum(P_w),1));
    end

    %The final estimate is some metric of these final resampling, such as
    %the mean value or variance
    x_est = mean(x_P);

    %{
    %the after
    subplot(133)
    plot(0,x_P_update,'.k','markersize',5)
    hold on
    plot(0,x_P,'.r','markersize',5)
    plot(0,x_est,'.g','markersize',40)
    xlabel('fixed time point')
    title('weight based resampling')
    pause
    %}
    % Save data in arrays for later plotting
    x_out = [x_out x];
    z_out = [z_out z];
    x_est_out = [x_est_out x_est];

end

t = 0:T;
figure(1);
clf
plot(t, x_out, '.-b', t, x_est_out, '-.r','linewidth',3);
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('time step'); ylabel('Quail flight position');
legend('True  position', 'Particle filter estimate');
figure(2)
plot(t,x_out-x_est_out,'b')
xlabel('time step'); ylabel('the error');
legend('the estimation error')
```

```

```

```python


from manifpy import SE2, SE2Tangent
import matplotlib
import matplotlib.pyplot as plt
# %matplotlib inline
import numpy as np
from numpy.linalg import inv


Vector = np.array


def Covariance():
    return np.zeros((SE2.DoF, SE2.DoF))


def Jacobian():
    return np.zeros((SE2.DoF, SE2.DoF))


if __name__ == '__main__':

    # START CONFIGURATION

    NUMBER_OF_LMKS_TO_MEASURE = 3 # to change back to 3

    # Define the robot pose element and its covariance
    X_simulation = SE2.Identity()
    X = SE2.Identity()

    X_unfiltered = SE2.Identity()
    P = Covariance()

    u_nom = Vector([0.1, 0.0, 0.05])
    u_sigmas = Vector([0.1, 0.1, 0.1])
    U = np.diagflat(np.square(u_sigmas))

    # Declare the Jacobians of the motion wrt robot and control
    J_x = Jacobian()
    J_u = Jacobian()

    # Define five landmarks in R^2
    landmarks = []
    landmarks.append(Vector([2.0,  0.0]))
    landmarks.append(Vector([2.0,  1.0]))
    landmarks.append(Vector([2.0, -1.0]))
    landmarks.append(Vector([2.0,  2.0]))
    landmarks.append(Vector([2.0,  4.0]))

    # Define the beacon's measurements
    measurements = [Vector([0, 0])] * NUMBER_OF_LMKS_TO_MEASURE

    y_sigmas = Vector([0.01, 0.01])
    R = np.diagflat(np.square(y_sigmas))

    # Declare some temporaries
```

```
    J_xi_x = Jacobian()
    J_e_xi = np.zeros((SE2.Dim, SE2.DoF))

    # CONFIGURATION DONE

    # pretty print
    np.set_printoptions(precision=3, suppress=True)

    # DEBUG
    print('X STATE     :    X      Y      Z    TH_x   TH_y   TH_z ')
    print('-------------------------------------------------------')
    print('X initial    : ', X_simulation.log().coeffs())
    print('X_est initial    : ', X.log().coeffs())
    print('-------------------------------------------------------')
    # END DEBUG

    # START TEMPORAL LOOP

    # com = Vector([1,2,5])
    # com_tg = SE2Tangent(com)
    # X = X.plus(com_tg, J_x, J_u)

    # building the desired state
    X_des = SE2.Identity()
    v_des = Vector([5, 0, 1.57])   # the desired position is [1,0] with
   orientation [pi/2]
    v_des_hat = SE2Tangent(v_des)
    X_des = X + v_des_hat
    kp = 0.02  #defining the gain kp

    # for ploting
    XX_des = np.array([]) # desired x
    XX = np.array([])
    XX_est = np.array([])
    XX_unf = np.array([])
    YY_des = np.array([]) # desired y
    YY = np.array([])
    YY_est = np.array([])
    YY_unf = np.array([])
    Th_des = np.array([]) # desired Theta
    Th = np.array([])
    Th_est = np.array([])
    Th_unf = np.array([])
    u_test = Vector([5, 2.0, 1.04])
    UUU = SE2Tangent(u_test)
    X = X + UUU


    # Make 10 steps. Measure up to three landmarks each time.
    for t in range(10):
        # I. Simulation
        # this is all for ploting later  **********************

        # X
        L = X_simulation.log().coeffs().transpose()
        XX = np.append(XX,L[0])
        YY =np.append(YY,L[1])
```

```
        Th =np.append(Th,L[2])
        print("here is the initial estimate:")
        print(X)
        print("hello there")
        NN =  X.log().coeffs().transpose()
        print(NN)
        # X_est
        L_est = X.log().coeffs().transpose()
        XX_est = np.append(XX_est,L_est[0])
        YY_est =np.append(YY_est,L_est[1])
        Th_est =np.append(Th_est,L_est[2])

        L_unf = X_unfiltered.log().coeffs().transpose()
        XX_unf = np.append(XX_unf,L_unf[0])
        YY_unf =np.append(YY_unf,L_unf[1])
        Th_unf =np.append(Th_unf,L_unf[2])

        # L_des = X_des.log().coeffs().transpose()    # for the desired trajectory
        # XX_des = np.append(XX_des,L_des[0])
        # YY_des =np.append(YY_des,L_des[1])
        # Th_des =np.append(Th_des,L_des[2])

        # Plotting ends here**********************

        # simulate noise

        u_noise = u_sigmas * np.random.rand(SE2.DoF)        # control noise
        u_noisy = u_nom + u_noise                           # noisy control

        u_simu = SE2Tangent(u_nom)
        u_est = SE2Tangent(u_noisy)
        u_unfilt = SE2Tangent(u_noisy)        # control noise u_noise = u_sigmas * np.random.rand(SE2.DoF)
        print(type(u_nom))
        print(type(u_noise))
        print(type(Vector(u_nom)))
        # u_noisy = np.array([u_nom])  + Vector(u_noise)                                  # noisy control u_noisy = u_nom + u_noise
        print("THE RESEARCHED u_nom TYPE IS FINALLY HERE :************")
        print(type(u_nom))
        # u_simu = SE2Tangent(Vector(u_nom))
        # u_est = SE2Tangent(Vector(u_nom))
        # u_unfilt = SE2Tangent(Vector(u_nom))
        print("THE RESEARCHED TYPE IS FINALLY HERE :************")
        print(type(u_simu))
        # first we move
        X_simulation = X_simulation + u_simu                # overloaded X.rplus(u) = X * exp(u)

        # then we measure all landmarks
        for i in range(NUMBER_OF_LMKS_TO_MEASURE):
            b = landmarks[i]                                # lmk coordinates in world frame

            # simulate noise
            y_noise = y_sigmas * np.random.rand(SE2.Dim)    # measurement
```

```
noise

        y = X_simulation.inverse().act(b)                 # landmark measurement, before adding noise

        y = y + y_noise                                   # landmark measurement, noisy
        measurements[i] = y                               # store for the estimator just below

    # II. Estimation

    # First we move

    X = X.plus(u_est, J_x, J_u)                           # X * exp(u), with Jacobians

    P = J_x * P * J_x.transpose() + J_u * U * J_u.transpose()

    # Then we correct using the measurements of each lmk
    for i in range(NUMBER_OF_LMKS_TO_MEASURE):
        # landmark
        b = landmarks[i]                                  # lmk coordinates in world frame

        # measurement
        y = measurements[i]                               # lmk measurement, noisy

        # expectation
        e = X.inverse(J_xi_x).act(b, J_e_xi)              # note: e = R.tr * ( b - t ), for X = (R,t).
        H = J_e_xi @ J_xi_x                               # Jacobian of the measurements wrt the robot pose. note: H = J_e_x = J_e_xi * J_xi_x
        E = H @ P @ H.transpose()

        # innovation
        z = y - e
        Z = E + R

        # Kalman gain
        K = P @ H.transpose() @ inv(Z)                    # K = P * H.tr * ( H * P * H.tr + R).inv

        # Correction step
        dx = K @ z                                        # dx is in the tangent space at X

        # Update
        X = X + SE2Tangent(dx)                            # overloaded X.rplus(dx) = X * exp(dx)
        P = P - K @ Z @ K.transpose()

    # III. Unfiltered

    # move also an unfiltered version for comparison purposes
    X_unfiltered = X_unfiltered + u_unfilt
```

```

        # IV. Results

        # DEBUG
        print('X simulated : ', X_simulation.log().coeffs().transpose())

        print(X_simulation)
        print('X estimated : ', X.log().coeffs().transpose())



        print('X unfilterd : ', X_unfiltered.log().coeffs().transpose())
        print('X desired : ', X_des.log().coeffs().transpose())

        print('-------------------------------------------------------')

        # u_simu = kp*X_des.lminus(X)
        # print("THE TYPE IS : ************************")
        # print(type(u_nom))
        # print("here you can see the type:")
        # print(type(u_nom))
        # u_nom = Vector(u_nom)
        # print(type(u_nom))
        # print(u_nom)
        # END DEBUG
    tt = np.arange(0,1,0.1)
    print(np.size(tt))
    print(np.size(XX))
    #PLOTING
#the x
    plt.plot(tt,XX,label = "X : True_Position")
    plt.plot(tt,XX_est,'r--',label = "X_est: Estimated_position")
    plt.plot(tt,XX_unf,'g',label = "X_unf: Unfiltered_Position")
    # plt.plot(tt,XX_des,'k',label = "X_des: desired position and orientation")
    plt.legend()
    plt.title("The difference between actual,estimated and unfiltered x position")
    plt.xlabel("Time")
    plt.ylabel("The position")
    # plt.figure(0)
    plt.show()
    #the e_x
    plt.figure(1)
    plt.plot(tt,XX-XX_est,'r--',label = "e_x")
    plt.legend()
    plt.title("the estimation error : X - X_est")
    # plt.plot(tt,XX-XX_des,'k--')
    plt.xlabel("Time")
    plt.ylabel("The x_error")
    plt.show()

# #the y
    plt.plot(tt,YY,label = "Y : True_Position")
    plt.plot(tt,YY_est,'r--',label = "Y_est: Estimated_position")
    plt.plot(tt,YY_unf,'g',label = "Y_unf: Unfiltered_Position")
    # plt.plot(tt,YY_des,'k',label = "Y_des: desired position and
```

```
    orientation")
    plt.legend()
    plt.title("The difference between actual,estimated and unfiltered y position")
    plt.xlabel("Time")
    plt.ylabel("The position")
    plt.figure(0)
    plt.show()
    print(5)
    #the e_y

    plt.figure(1)
    plt.plot(tt,YY-YY_est,'r--',label = "e_y")
    plt.legend()
    # plt.plot(tt,YY-YY_des,'k--')
    plt.title("the estimation error : Y - Y_est")
    plt.xlabel("Time")
    plt.ylabel("The y_error")
    plt.show()
# #the theta
    plt.plot(tt,Th,label = "Th : True_Position")
    plt.plot(tt,Th_est,'r--',label = "Th_est: Estimated_position")
    plt.plot(tt,Th_unf,'g',label = "Th_unf: Unfiltered_Position")
    # plt.plot(tt,Th_des,'k',label = "Th_des: desired position and orientation")
    plt.legend()
    plt.title("The difference between actual,estimated and unfiltered orientation theta")
    plt.xlabel("Time")
    plt.ylabel("The position")
    plt.figure(0)
    plt.show()
    print(5)
    #the e_th
    plt.figure(1)
    plt.plot(tt,Th-Th_est,'r--',label = 'e_theta')
    plt.legend()
    plt.title("the estimation error : Theta - Theta_est")
    # plt.plot(tt,Th-Th_des,'k--')
    plt.xlabel("Time")
    plt.ylabel("The th_error")
    plt.show()


    # print(dir(manifpy))
    # plt.show();

    # plt.plot(tt,X.log().coeffs().transpose());
    # plt.show();
```

```



from manifpy import SE2, SE2Tangent
import matplotlib
import matplotlib.pyplot as plt
# %matplotlib inline
```

```python
import numpy as np
from numpy.linalg import inv


Vector = np.array


def Covariance():
    return np.zeros((SE2.DoF, SE2.DoF))


def Jacobian():
    return np.zeros((SE2.DoF, SE2.DoF))


if __name__ == '__main__':

    # START CONFIGURATION

    NUMBER_OF_LMKS_TO_MEASURE = 3 # to change back to 3

    # Define the robot pose element and its covariance
    X_simulation = SE2.Identity()
    X = SE2.Identity()

    X_unfiltered = SE2.Identity()
    P = Covariance()

    u_nom = Vector([0.1, 0.0, 0.05])
    u_sigmas = Vector([0.1, 0.1, 0.1])
    U = np.diagflat(np.square(u_sigmas))

    # Declare the Jacobians of the motion wrt robot and control
    J_x = Jacobian()
    J_u = Jacobian()

    # Define five landmarks in R^2
    landmarks = []
    landmarks.append(Vector([2.0,  0.0]))
    landmarks.append(Vector([2.0,  1.0]))
    landmarks.append(Vector([2.0, -1.0]))
    landmarks.append(Vector([2.0,  2.0]))
    landmarks.append(Vector([2.0,  4.0]))

    # Define the beacon's measurements
    measurements = [Vector([0, 0])] * NUMBER_OF_LMKS_TO_MEASURE

    y_sigmas = Vector([0.01, 0.01])
    R = np.diagflat(np.square(y_sigmas))

    # Declare some temporaries
    J_xi_x = Jacobian()
    J_e_xi = np.zeros((SE2.Dim, SE2.DoF))

    # CONFIGURATION DONE

    # pretty print
```

```
    np.set_printoptions(precision=3, suppress=True)

    # DEBUG
    print('X STATE     :    X      Y      Z    TH_x   TH_y   TH_z ')
    print('-------------------------------------------------------')
    print('X initial    : ', X_simulation.log().coeffs())
    print('X_est initial    : ', X.log().coeffs())
    print('-------------------------------------------------------')
    # END DEBUG

    # START TEMPORAL LOOP

    # com = Vector([1,2,5])
    # com_tg = SE2Tangent(com)
    # X = X.plus(com_tg, J_x, J_u)

    # building the desired state
    X_des = SE2.Identity()
    v_des = Vector([5, 0, 1.57])   # the desired position is [1,0] with
   orientation [pi/2]
    v_des_hat = SE2Tangent(v_des)
    X_des = X + v_des_hat
    kp = 0.02  #defining the gain kp

    # for ploting
    XX_des = np.array([]) # desired x
    XX = np.array([])
    XX_est = np.array([])
    XX_unf = np.array([])
    YY_des = np.array([]) # desired y
    YY = np.array([])
    YY_est = np.array([])
    YY_unf = np.array([])
    Th_des = np.array([]) # desired Theta
    Th = np.array([])
    Th_est = np.array([])
    Th_unf = np.array([])
    u_test = Vector([5, 2.0, 1.04])
    UUU = SE2Tangent(u_test)
    X = X + UUU


    u_noise = u_sigmas * np.random.rand(SE2.DoF)          # control noise
    u_noisy = u_nom + u_noise                             # noisy control

    u_simu = SE2Tangent(u_nom)
    u_est = SE2Tangent(u_noisy)
    u_unfilt = SE2Tangent(u_noisy)
    # Make 10 steps. Measure up to three landmarks each time.
    for t in range(500):
        # I. Simulation
        # this is all for ploting later   **********************

        # X
        L = X_simulation.log().coeffs().transpose()
        XX = np.append(XX,L[0])
        YY =np.append(YY,L[1])
```

```
        Th =np.append(Th,L[2])
        print("here is the initial estimate:")
        print(X)
        print("hello there")
        NN =  X.log().coeffs().transpose()
        print(NN)
        # X_est
        L_est = X.log().coeffs().transpose()
        XX_est = np.append(XX_est,L_est[0])
        YY_est =np.append(YY_est,L_est[1])
        Th_est =np.append(Th_est,L_est[2])

        L_unf = X_unfiltered.log().coeffs().transpose()
        XX_unf = np.append(XX_unf,L_unf[0])
        YY_unf =np.append(YY_unf,L_unf[1])
        Th_unf =np.append(Th_unf,L_unf[2])

        L_des = X_des.log().coeffs().transpose()
        XX_des = np.append(XX_des,L_des[0])
        YY_des =np.append(YY_des,L_des[1])
        Th_des =np.append(Th_des,L_des[2])
        # Plotting ends here*********************

        # simulate noise
               # control noise u_noise = u_sigmas * np.random.rand(SE2.DoF)
        print(type(u_nom))
        print(type(u_noise))
        print(type(Vector(u_nom)))
        # u_noisy = np.array([u_nom])  + Vector(u_noise)
          # noisy control u_noisy = u_nom + u_noise
        print("THE RESEARCHED u_nom TYPE IS FINALLY HERE :*************")
        print(type(u_nom))
        # u_simu = SE2Tangent(Vector(u_nom))
        # u_est = SE2Tangent(Vector(u_nom))
        # u_unfilt = SE2Tangent(Vector(u_nom))
        print("THE RESEARCHED TYPE IS FINALLY HERE :*************")
        print(type(u_simu))
        # first we move
        X_simulation = X_simulation + u_simu                # overloaded X.rplus(u) = X * exp(u)

        # then we measure all landmarks
        for i in range(NUMBER_OF_LMKS_TO_MEASURE):
            b = landmarks[i]                                # lmk coordinates in world frame

            # simulate noise
            y_noise = y_sigmas * np.random.rand(SE2.Dim)    # measurement noise

            y = X_simulation.inverse().act(b)               # landmark measurement, before adding noise

            y = y + y_noise                                 # landmark measurement, noisy
            measurements[i] = y                             # store for the estimator just below
```

```

    # II. Estimation

    # First we move

    X = X.plus(u_est, J_x, J_u)                    # X * exp(u), with Jacobians

    P = J_x * P * J_x.transpose() + J_u * U * J_u.transpose()

    # Then we correct using the measurements of each lmk
    for i in range(NUMBER_OF_LMKS_TO_MEASURE):
        # landmark
        b = landmarks[i]                           # lmk coordinates in world frame

        # measurement
        y = measurements[i]                        # lmk measurement, noisy

        # expectation
        e = X.inverse(J_xi_x).act(b, J_e_xi)       # note: e = R.tr * ( b - t ), for X = (R,t).
        H = J_e_xi @ J_xi_x                        # Jacobian of the measurements wrt the robot pose. note: H = J_e_x = J_e_xi * J_xi_x
        E = H @ P @ H.transpose()

        # innovation
        z = y - e
        Z = E + R

        # Kalman gain
        K = P @ H.transpose() @ inv(Z)             # K = P * H.tr * ( H * P * H.tr + R).inv

        # Correction step
        dx = K @ z                                 # dx is in the tangent space at X

        # Update
        X = X + SE2Tangent(dx)                     # overloaded X.rplus(dx) = X * exp(dx)
        P = P - K @ Z @ K.transpose()

    # III. Unfiltered

    # move also an unfiltered version for comparison purposes
    X_unfiltered = X_unfiltered + u_unfilt

    # IV. Results

    # DEBUG
    print('X simulated : ', X_simulation.log().coeffs().transpose())

    print(X_simulation)
    print('X estimated : ', X.log().coeffs().transpose())

```

```
        print('X unfilterd : ', X_unfiltered.log().coeffs().transpose())
        print('X desired : ', X_des.log().coeffs().transpose())

        print('--------------------------------------------------------')

        u_simu = kp*X_des.lminus(X)
        # print("THE TYPE IS : *************************")
        # print(type(u_nom))
        # print("here you can see the type:")
        # print(type(u_nom))
        # u_nom = Vector(u_nom)
        # print(type(u_nom))
        # print(u_nom)
        # END DEBUG
    tt = np.arange(0,50,0.1)
    print(np.size(tt))
    print(np.size(XX))
    #PLOTING
#the x
    plt.plot(tt,XX,label = "X : True_Position")
    plt.plot(tt,XX_est,'r--',label = "X_est: Estimated_position")
    plt.plot(tt,XX_unf,'g',label = "X_unf: Unfiltered_Position")
    plt.plot(tt,XX_des,'k',label = "X_des: desired position and orientation
   ")
    plt.legend()
    plt.title("The difference between actual, estimated, unfiltered and
   desired x position")
    plt.xlabel("Time")
    plt.ylabel("The position")
    # plt.figure(0)
    plt.show()
    #the e_x
    plt.figure(1)
    plt.plot(tt,XX-XX_est,'r--',label = 'the estimation error ')
    plt.plot(tt,XX-XX_des,'k--',label = 'the regulation error ')
    plt.legend()
    plt.xlabel("Time")
    plt.ylabel("The x_error")
    plt.show()

# #the y
    plt.plot(tt,YY,label = "Y : True_Position")
    plt.plot(tt,YY_est,'r--',label = "Y_est: Estimated_position")
    plt.plot(tt,YY_unf,'g',label = "Y_unf: Unfiltered_Position")
    plt.plot(tt,YY_des,'k',label = "Y_des: desired position and orientation
   ")
    plt.legend()
    plt.title("The difference between actual,estimated, unfiltered and
   desired  y position")
    plt.xlabel("Time")
    plt.ylabel("The position")
    plt.figure(0)
    plt.show()
    print(5)
    #the e_y
```

```
    plt.figure(1)
    plt.plot(tt,YY-YY_est,'r--',label = "the estimation error")
    plt.plot(tt,YY-YY_des,'k--',label = "the regulation error")
    plt.legend()
    plt.xlabel("Time")
    plt.ylabel("The y_error")
    plt.show()
# #the theta
    plt.plot(tt,Th,label = "Th : True_Position")
    plt.plot(tt,Th_est,'r--',label = "Th_est: Estimated_position")
    plt.plot(tt,Th_unf,'g',label = "Th_unf: Unfiltered_Position")
    plt.plot(tt,Th_des,'k',label = "Th_des: desired position and orientation")
    plt.legend()
    plt.title("The difference between actual,estimated, unfiltered and desired orientation theta")
    plt.xlabel("Time")
    plt.ylabel("The position")
    plt.figure(0)
    plt.show()
    print(5)
    #the e_th
    plt.figure(1)
    plt.plot(tt,Th-Th_est,'r--',label = "the estimation error")
    plt.plot(tt,Th-Th_des,'k--',label = "the regulation error")
    plt.legend()
    plt.xlabel("Time")
    plt.ylabel("The th_error")
    plt.show()


    # print(dir(manifpy))
    # plt.show();

    # plt.plot(tt,X.log().coeffs().transpose());
    # plt.show();
```

Bibliography

[1] J. Sola, J. Deray, and D. Atchuthan, “A micro lie theory for state estimation in robotics,” *arXiv preprint arXiv:1812.01537*, 2018.

[2] G. C. G. Cortés, F. Castanos, and J. Dávila, “Sliding motions on so (3), sliding subgroups,” in *2019 IEEE 58th Conference on Decision and Control (CDC)*, IEEE, 2019, pp. 6953–6958.

[3] R. Howe, “Very basic lie theory,” *The American Mathematical Monthly*, vol. 90, no. 9, pp. 600–623, 1983.

[4] J. Stillwell, *Naive lie theory.* Springer Science & Business Media, 2008.

[5] E. Eade, “Lie groups for 2d and 3d transformations,” *URL http://ethaneade. com/lie. pdf, revised Dec*, vol. 117, p. 118, 2013.

[6] S. Fiori, “Manifold calculus in system theory and control—fundamentals and first-order systems,” *Symmetry*, vol. 13, no. 11, p. 2092, 2021.

[7] T. D. Barfoot, *State estimation for robotics.* Cambridge University Press, 2017.

[8] Y. Yu, S. Yang, M. Wang, C. Li, and Z. Li, “High performance full attitude control of a quadrotor on so (3),” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 1698–1703.

[9] G. N. P. Pratama, A. I. Cahyadi, and S. Herdjunanto, “Robust proportional-derivative control on so (3) for transporting quadrotor with load uncertainties,” *IAENG International Journal of Computer Science*, vol. 45, no. 2, pp. 275–284, 2018.

[10] F. Bullo and R. M. Murray, “Proportional derivative (pd) control on the euclidean group,” 1995.

[11] V. Utkin, “Variable structure systems with sliding modes,” *IEEE Transactions on Automatic control*, vol. 22, no. 2, pp. 212–222, 1977.

[12] A. Saccon, J. Hauser, and A. P. Aguiar, “Optimal control on lie groups: The projection operator approach,” *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2230–2245, 2013.

[13] A. Torgesen, “Optimal linear control on the so (2) manifold using lie algebras and auto-differentiation,”

[14] P. Coelho and U. Nunes, “Lie algebra application to mobile robot control: A tutorial,” *Robotica*, vol. 21, no. 5, pp. 483–493, 2003.

[15] V. Jurdjevic, J. Velimir, and V. Đurđević, *Geometric control theory.* Cambridge university press, 1997.

[16] H. A. Hashim, "Special orthogonal group so (3), euler angles, angle-axis, rodriguez vector and unit-quaternion: Overview, mapping and challenges," *arXiv preprint arXiv:1909.06669*, 2019.

[17] H. Eslamiat, N. Wang, and A. K. Sanyal, "Geometric pid-type attitude tracking control on so (3)," *arXiv preprint arXiv:1909.06916*, 2019.

[18] D. E. Catlin, *Estimation, control, and the discrete Kalman filter*. Springer Science & Business Media, 2012, vol. 71.

[19] G. Welch, G. Bishop, *et al.*, "An introduction to the kalman filter," 1995.

[20] G. Bourmaud, R. Mégret, A. Giremus, and Y. Berthoumieu, "Discrete extended kalman filter on lie groups," in *21st European Signal Processing Conference (EUSIPCO 2013)*, IEEE, 2013, pp. 1–5.

[21] M. S. Grewal and A. P. Andrews, *Kalman filtering: Theory and Practice with MATLAB*. John Wiley & Sons, 2014.

[22] M. Bosse, G. Agamennoni, I. Gilitschenski, *et al.*, "Robust estimation and applications in robotics," *Foundations and Trends® in Robotics*, vol. 4, no. 4, pp. 225–269, 2016.

[23] T. Hatanaka, N. Chopra, M. Fujita, and M. W. Spong, *Passivity-based control and estimation in networked robotics*. Springer, 2015.

[24] D. Simon, *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[25] J. Ćesić, I. Marković, and I. Petrmović, "Moving object tracking employing rigid body motion on matrix lie groups," in *2016 19th International Conference on Information Fusion (FUSION)*, IEEE, 2016, pp. 2109–2115.

[26] A. H. de Ruiter, "So (3)-constrained kalman filtering with application to attitude estimation," in *2014 American Control Conference*, IEEE, 2014, pp. 4937–4942.

[27] G. Marjanovic and V. Solo, "An engineer's guide to particle filtering on matrix lie groups," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2016, pp. 3969–3973.

[28] J. Elfring, E. Torta, and R. van de Molengraft, "Particle filters: A hands-on tutorial," *Sensors*, vol. 21, no. 2, p. 438, 2021.

[29] T. Yang, P. G. Mehta, and S. P. Meyn, "Feedback particle filter," *IEEE transactions on Automatic control*, vol. 58, no. 10, pp. 2465–2480, 2013.

[30] H. Niu and Z. Geng, "Stabilisation of a relative equilibrium of an underactuated auv on se (3)," *International Journal of Control*, vol. 92, no. 8, pp. 1883–1902, 2019.

[31] M. L. Seto, *Marine robot autonomy*. Springer Science & Business Media, 2012.

[32] H. C. Henninger, K. D. von Ellenrieder, and J. D. Biggs, "Trajectory generation and tracking on se (3) for an underactuated auv with disturbances," *IFAC-PapersOnLine*, vol. 52, no. 21, pp. 242–247, 2019.

[33] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*, IEEE, 2010, pp. 5420–5425.

[34] J. D. Lawson and Y. Lim, “The geometric mean, matrices, metrics, and more,” *The American Mathematical Monthly*, vol. 108, no. 9, pp. 797–812, 2001.

[35] J. Lawson, “Matrix lie groups and control theory,” 2007.

[36] G. S. Chirikjian, *Stochastic models, information theory, and Lie groups, volume 2: Analytic methods and modern applications*. Springer Science & Business Media, 2011, vol. 2.

[37] M. Andonian, D. Cazzaro, L. Invernizzi, M. Chyba, and S. Grammatico, “Geometric control for autonomous underwater vehicles: Overcoming a thruster failure,” in *49th IEEE Conference on Decision and Control (CDC)*, IEEE, 2010, pp. 7051–7056.

[38] E. Campos, A. Chemori, V. Creuze, J. Torres, and R. Lozano, “Saturation based nonlinear depth and yaw control of underwater vehicles with stability analysis and real-time experiments,” *Mechatronics*, vol. 45, pp. 49–59, 2017.

[39] R. A. Chavan, T. Seigler, and J. B. Hoagg, “Somall-satellite attitude consensus on so (3) using continuous but only piecewise-continuously differentiable sinusoidal controls,” in *2021 American Control Conference (ACC)*, IEEE, 2021, pp. 4255–4260.

[40] R. Ortega and E. Garcia-Canseco, “Interconnection and damping assignment passivity-based control: A survey,” *European Journal of control*, vol. 10, no. 5, pp. 432–450, 2004.

[41] R. Yang, “Modeling and robust control approach for autonomous underwater vehicles,” Ph.D. dissertation, Université de Bretagne occidentale-Brest; Zhongguo hai yang da xue (Qingdao ..., 2016.

[42] T. Liu, Y. Hu, and H. Xu, “Deep reinforcement learning for vectored thruster autonomous underwater vehicle control,” *Complexity*, vol. 2021, 2021.