Doctoral Thesis Reviewed
by Ritsumeikan University

Environment-Aided Manipulation (EAM) with Snake Robots
(ヘビ型ロボットにおける環境支援型マニピュレーション)

September 2018
２０１８年９月

Doctoral Program in Advanced Mechanical Engineering and Robotics
Graduate School of Science and Engineering
Ritsumeikan University

立命館大学大学院理工学研究科
機械システム専攻博士課程後期課程

REYES PINNER FABIAN EUGENIO
レイエス　ピナー　ファビアン　エウヘニオ

Supervisor: Professor MA Shugen
研究指導教員:　馬　書根　教授

Fall 2018

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

Committee in Charge:

- Professor MA Shugen, Chair
- Professor HIRAI Shinichi
- Professor NOKATA Makoto

Declaration of Authorship

I, Fabian Eugenio REYES PINNER, declare that this thesis titled, "Environment-Aided Manipulation (EAM) with Snake Robots" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"All men dream: but not equally. Those who dream by night in the dusty recesses of their minds, wake in the day to find that it was vanity: but the dreamers of the day are dangerous men, for they may act their dream with open eyes, to make it possible. This, I did."

T. E. Lawrence

RITSUMEIKAN UNIVERSITY

Abstract

Graduate School of Science and Engineering

Department of Robotics

Doctor of Philosophy

Environment-Aided Manipulation (EAM) with Snake Robots

by Fabian Eugenio REYES PINNER

In this thesis, a snake robot in contact with an object is analyzed. The intention is to use the snake robot to grasp and manipulate the object. This is done by studying this system under the framework of multi-rigid-body systems, which is compatible with other research regarding manipulation or even walking with robotic systems.

The main contribution of this thesis is the additional study of the interaction of the snake robot with the environment. A snake robot is normally in contact with the environment, for example, laying on the ground. Unlike robotic manipulators that avoid this contact, snake robots may not have another choice. By understanding this interaction with the environment, we can understand how to exploit it, or at least alleviate its influence. This is where the concept of Environment-aided Manipulation (EAM) is born.

EAM is intended to study any robotic system that tries to manipulate an object, but simultaneously has contact with the environment (or any other external object). We will study what parameters of the snake robot and object have a significant impact on the performance of the manipulation task.

When trying to manipulate an object with a snake robot, the most important point is to answer *feasibility*. In other words, it is important to know if it is even possible to move the object. This is a question that has not been answered in previous research of snake robots, due to the fact that often inertial terms are ignored. But we will show that the inertial terms of the snake robot and object play an important role, they cannot be ignored.

First, the relationship between the size of the snake robot and object is studied to determine if it is possible to grasp it. This shows the similarities of snake robots with other robotic systems like manipulators, or even robotic hands, since the kinematic structure is similar. Later, sufficient requirements to grasp the object with form-closure are presented, based on a geometrical analysis.

Then, a complete study of the subspaces involved is presented. This is done in the framework of robotic grasping, which is a well-studied framework with several applications, but that had not been extended to snake robots previously. The projections between joint torques of the snake robot and the resulting forces applied to the object are studied, and conditions to guarantee form-closure are presented.

Once a complete analysis of the dynamical model of the snake robot and object are presented, an analysis using basic differential geometry is shown. This analysis guarantees invariance of the quantities due to a change of coordinates. In other words, the analysis presented does not depend on a specific frame of reference. The analysis considers inertial terms to consider the feasibility due to the inertial parameters of robot and object, which is something that cannot be understood purely from a kinematic analysis. In other words, a object that is too heavy could not be moved, regardless of the control strategy.

It is shown that there is a set of postures (shape of the snake robot) that are optimal for pushing the object. More specifically, it is shown that the position of the center of mass of the snake robot

plays a paramount role when pushing an object. This is regardless of the specific joints' configurations. In other words, several postures with different joint coordinates will have a similar performance. It is also shown that the friction between the snake robot and ground has almost no impact on the instantaneous performance. In other words, the analysis presented shows that friction can be ignored when calculating optimal postures. This is even more important for unstructured environments, where determining the friction coefficient would be difficult.

Then, the analysis is extended to consider also the motion of the system. Intuitively, it can be understood that if pushing against an object, it is desired that the robot does not alter its posture significantly due to the interaction. It is shown that the postures that maximizes the motion of the object to be manipulated, are not necessarily the same postures that minimize the motion of the snake robot. This implies a trade off in the control strategy, a conclusion that could not have been drawn without the previous analysis.

Finally, a rigorous set of experiments is perform to validate the mathematical analysis presented. The complete description of the experiment setup, prototype, and results is shown.

Acknowledgements

This thesis could not have been possible without the financing and support from the Ministry of Education, Culture, Sports, Science and Technology-Japan (MEXT), whose financing for the first three years allowed for several publications and participation in conferences. I also would like to thank all the Ritsumeikan University staff that helped me with the necessary procedures and everyday life problems that arose during my stay in here.

I give personal thanks to all the members of the BioInMech Laboratory for their support and intellectual discussions that helped me to improve my research. In particular, thanks to my supervisor Dr. Shugen Ma that was always very patient with me. Thanks to Dr. Kakogawa Atsushi, Dr. Tian Yang, and Dr. Matsuno Takahiro for their support and guidance during my stay here. Also thanks to M. Eng. Matsumoto Hisanori for his help in my research. Thanks to everyone for for their friendship and help. I apologize to everyone I did not mention.

Finally, thanks to my family for the continuous support. Without their caring and encouragement, it would not have been possible to conclude achieve this. I regret not thanking them enough. Thanks to my father Daniel Reyes for his continuous encouragement, which I have never deserved. Thanks to my sister for her help, although sometimes in the form of reprimands, never ill-intended. Also thanks to my aunt Alicia Cabañas, whose support was crucial for the fulfillment of this milestone.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

AB	**A**rticulated-**b**ody
COM	**C**enter **o**f **M**ass
CRB	**C**omposite-**r**igid-**b**ody
DOF	**D**egrees **o**f **F**reedom
EAM	**E**nvironment-**a**ided **M**anipulation
OAL	**O**bstacle-**a**ided **L**ocomotion
OPPAS	**OP**en-source **PA**rametric **S**nake-robot
SC	**S**erpenoid **C**urve
sr	slippage ratio

List of Symbols

Symbol	Name	Unit
$\boldsymbol{a}, \boldsymbol{b}, \cdots$	Vectors will be denoted with lower-case bold letters	
$\boldsymbol{A}, \boldsymbol{B}, \cdots$	Matrices will be denoted with upper-case bold letters	
$\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \cdots$	unitary vectors (usually spanning vectors) have a hat	
n_{op}	Dimension of operational space twists/wrenches	
n_s	Degrees of freedom of the snake robot (including floating base)	
n_{na}	Number of non-actuated degrees of freedom of the snake robot (floating base)	
n_a	Number of actuated joints of the snake robot	
n_ℓ	Number of links of the snake robot ($n_\ell = n_a + 1$)	
n_{obj}	Degrees of freedom of the object	
n	Degrees of freedom of the snake robot + object(s) (i.e., $n = n_s + n_{obj}$)	
$n_{c,f}$	Number of constraints due to (static) friction	
$n_{c,np}$	Number of non-penetration constraints	
n_c	Total number of constraints (i.e., $n_c = n_{c,f} + n_{c,np}$)	
m_s	mass of one link of the snake robot	kg
m_T	total mass of the snake robot (i.e., $m_T = n_\ell m_s$)	kg
m_{obj}	mass of object being pushed by snake robot	kg
κ	ratio between mass of object and snake robot's link (i.e., $\kappa := m_{obj}/m_s$)	
r_{obj}	radio of circle circumscribed to an object to be manipulated	meter
ℓ	length of a link of the snake robot	meter
w	width of a link of the snake robot	meter
$\boldsymbol{M}_s \in \Re^{n_s \times n_s}$	Inertia matrix of the snake robot	
$\boldsymbol{I}_{obj} \in \Re^{n_{obj} \times n_{obj}}$	Inertia matrix of an object	
m_1	mass denoting the mass of a generic system numbered '1'	kg
m_2	mass denoting the mass of a generic system numbered '2'	kg
κ	dimensionless ratio relating masses of two systems as $m_2 = m_1\kappa$	
$\boldsymbol{q}_{na} \in \Re^{n_{na}}$	non-actuated DOFs of the snake robot	
$\boldsymbol{q}_a \in \Re^{n_a}$	actuated DOFs of the snake robot (i.e., joints)	
$\boldsymbol{q}_s \in \Re^{n_s}$	generalized coordinates of the snake robot (i.e., $\boldsymbol{q}_s = [\boldsymbol{q}_{na}^T + \boldsymbol{q}_a^T]^T$)	
$\boldsymbol{q}_{obj} \in \Re^{n_{obj}}$	state of the object(s)	
$\boldsymbol{q} \in \Re^n$	state of whole system (i.e., $\boldsymbol{q} = [\boldsymbol{q}_s^T + \boldsymbol{q}_{obj}^T]^T$)	
$\mathrm{F}^n \in \Re^n$	n-dimensional subspace of generalized forces	
$\mathrm{M}^n \in \Re^n$	n-dimensional subspace of generalized motions	
$\mathcal{C} \in \mathrm{F}^n$	n_c-dimensional subspace of constraint forces embedded in F^n	
$\mathcal{D} \in \mathrm{M}^n$	$(n - n_c)$-dimensional subspace unconstrained motions embedded in M^n	
$\boldsymbol{T}_b \in R^{n_{op}}$	Spanning vectors (wrenches) of non-penetration constraint forces	
$\boldsymbol{T}_{b,n}$	Spanning vectors (moments) of non-penetration constraint forces	

$\boldsymbol{T}_{b,f}$	Spanning vectors (forces) of non-penetration constraint forces
$\boldsymbol{T}_f \in R^{n_{op}}$	Spanning vectors (wrenches) of friction constraint forces
$\boldsymbol{T}_{f,n}$	Spanning vectors (moments) of friction constraint forces
$\boldsymbol{T}_{f,f}$	Spanning vectors (forces) of friction constraint forces
$\boldsymbol{f}_c \in \mathrm{F}^n$	constraint forces
$\boldsymbol{f}_{c,np}$	non-penetration constraint forces
$\boldsymbol{f}_{c,f}$	constraint forces due to friction
$\boldsymbol{\lambda} \in \mathcal{C}$	constraint forces (coordinates)
$\boldsymbol{\lambda}_{np} \in \Re^{n_{c,np}}$	non-penetration constraint forces (coordinates)
$\boldsymbol{\lambda}_b \in \Re^{n_{c,np}}$	non-penetration constraint forces (coordinates). Equivalent to $\boldsymbol{\lambda}_{np}$
$\boldsymbol{\lambda}_f \in \Re^{n_{c,f}}$	tuple of scalars representing constraint forces due to friction (coordinates)
$\boldsymbol{a} \circ \boldsymbol{b}$	Scalar product between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$\boldsymbol{a} \times \boldsymbol{b}$	Cross product between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$a \leqslant b$	Inequalities between scalars
$\boldsymbol{a} \leqslant \boldsymbol{b}$	Inequalities between vectors. Interpreted element-wise
$< \cdot, \cdot >$	Inner product
$\boldsymbol{a}^\times$	cross-product operator (also called skew-operator) of a vector $\boldsymbol{a}$
$\|\|\boldsymbol{a}\|\|$	Norm of a vector a. Not necessarily the Euclidean norm
${}^B\boldsymbol{X}_A \in \Re^{6\times 6}$	Transformation (for twists) from frame A to B
${}^B\boldsymbol{X}_A^* \in \Re^{6\times 6}$	Transformation (for wrenches) from frame A to B

Dedicado a mi familia . . .

Chapter 1

Introduction

Snake robots have shown a lot of promise for locomotion in unstructured environments. However their application in other tasks, for example manipulating objects, is yet to be deeply studied. The intention of this thesis is study snake robots in tasks related to manipulation and grasping, while doing it in a framework that is consistent with other robotic systems.

Snake robots, although similar to robotic manipulators in their kinematic structure, are more similar to mobile robots due to their lack of a fixed-base. The proper framework to study them depends also on the task to be performed. If a snake robot is used to interact with objects, for example to grab them and manipulate them, then the objects' dynamics have to be considered in the modeling and control stages. In addition, the snake robot is interacting with the environment (e.g., touching the ground with its belly to obtain propulsive forces) and this interaction has to be accounted for.

We propose that the interaction with the environment can be studied along the interaction with an object in the framework of *multibody rigid-body systems*. This would put snake robots in a framework consistent with other mobile robots at the same time as grasping and manipulation with robotic hands, for example.

The interaction between the snake robot and the environment has a deep effect on the interaction between the snake robot and object, as it will be studied. Since the snake robot is a mobile robot, its inertial effects cannot be simply dismissed without further study. The mass of the snake robot and its shape will also influence the tasks that it can perform.

1.1 Literature Review

Locomotion and manipulation have been long-time studied topics in robotics. The ability to locomote in places or manipulate objects that humans cannot is an attractive proposal that has deep social and economical repercussions. However, real life applications may require robots that are multi-purpose. It is necessary that the robot can adapt to several situations and changes in its environment. For example, mobile robots may encounter situations where they would

need to interact with the environment or manipulate an object. On the other hand, robots specifically built for manipulation (e.g., robotic arms) are usually used in structured environments [1]. This limits their usability and adaptability. Therefore, robots that are capable of both locomotion and manipulation are a natural and logical extension of robotic research.

Locomotion can be achieved by several mechanical means. For example, robots with wheels, treads, or legs are capable of locomotion, each with its unique advantages and disadvantages. Manipulation and grasping using robots has been one of the most heavily researched area of robotics, mainly due to its potential applications in the manufacturing industry. Several types of mechanisms have been designed and researched to study manipulation and grasping, including mechanisms based on human arms and hands.

The combination of both locomotion and manipulation can be achieved by combining both types of systems. The simplest idea is to have a mobile base and add a robotic arm [1, 2]. The main drawback is that the robot's capabilities will be determined by the mechanical design of each part. Another problem is that this means the robot is divided, at least conceptually, into two parts: the locomotion part, and the manipulation part. This will greatly influence the control strategy, since each part will usually have its own modeling and control approach. The main line of thought of this type of research is *how can the locomotion improve the manipulation*, or vice versa.

Snake robots are robots that have a structure similar to biological snakes. This type of robot has gained attention in the past few years, mainly due to its unique locomotion gaits, which show a lot of promise for improving locomotion in unstructured environments. To be able to move in unstructured environments is an attractive idea that has several applications.

For example, in Search and Rescue (SAR) operations [3, 4] more common mobile robots have show limitations. Snake robots have been proposed as an improvement [5]. Environmental monitoring could also benefit from robots that can mimic their biological counterparts [6].

Locomotion in planar environments has been probably the main topic of research for snake robots [7, 8, 9]. Research has been extended to motion in planar slopes [10, 11], and motion in 3D-space [12, 13] Also, there are and more broad studies on locomotion [14] using snake robots.

However, snake robots should not be limited to only locomotion tasks. Although snake robots could excel in locomotion, with the current stage of research it is not clear if they can be used to interact with the environment (or an object) dexterously. For example, a snake robot could try to grab and manipulate an object, or at least push it in a desired trajectory. A snake robot's (kinematic) structure resembles a robotic manipulator but there are key differences that have not been fully addressed in previous research and will be discussed more deeply in Section 2.1. The first key point is that a snake robot is a mobile robot; it has a floating base making it an underactuated system. Unlike a robotic manipulator with a fixed-base, the snake robot is a mobile robot that may slip while interacting with an object, limiting the set of forces that it can

[1] By **structured** we mean an environment that is well known, and in general is static (there are no moving obstacles, and if they are, its state can be known *a priori*). Additionally, its geometry can be explained with geometric equations (e.g., a plane, boolean combinations of basic primitive shapes).

produce. Another key difference is that snake robots may contact the environment at several points of its body. Snake robots rely on the friction between its belly and the ground to obtain propulsive forces for locomotion and may be contacting the environment (e.g., a wall) at the same time.

Because the kinematic structure of a snake robot resembles a robotic arm, papers that deal with similar situations can be found in existing literature. In [15] the manipulability of a tentacle arm (a robotic arm consisting of several links) manipulating an object was studied. In [16] a hyper-redundant serial robot was considered and both locomotion and manipulation of an object were considered. In both cases the analysis was purely kinematic while assuming a fixed-base robotic system. Also, there was no force analysis between the robotic arms and object or with the environment. In other words, the main features of a snake robot (i.e., lack of a fixed-base and extra interaction with the environment) were either ignored or over simplified. In [17] the duality between locomotion and manipulation of a snake robot was considered under the assumption that the snake robot can be treated similarly to a robotic arm with a fixed-base when manipulating an object. In [18] a snake robot is used to grasp and drag an object. However, there is no force analysis (either quasi-static of dynamic) of the interaction between the snake robot and environment or between the snake robot and grasped object (i.e., the analysis is purely based on a kinematic model similar to a robotic manipulator). More recently, the idea of attaching a saw to a snake robot and using it for a cutting task has been proposed [19]. A dynamic model and control law were proposed showing that a snake robot can be used for tasks other than pure locomotion.

An interesting idea that combines locomotion and interaction with the environment, called obstacle-aided locomotion (OAL), has been proposed in [20] where obstacles in the environment are used as auxiliary sources for propulsion (or to avoid jamming). However, the focus is still on locomotion and not in controlling the contact forces with the obstacles. Extending the concept of OAL, in [21, 22] a snake robot pushes against rigid objects in order to move in a certain direction. However, the analysis is limited to just an instant in time and is difficult to confirm if this approach could be used as a locomotive gait. In [23] the interaction of a snake-like robot and obstacles is also studied, which is a similar idea to OAL. From the control point of view, a strategy for calculating the torque input in order to move the snake robot towards a desired point via convex optimization was presented in [24]. However, one of the key findings was that it is extremely difficult for a snake robot to be controlled like a robotic manipulator since the friction between the ground and the snake robot is not big enough to prevent slipping of the floating base (contradicting the assumption of a fixed-based used in previous research), but big enough to not be negligible. Since friction is related to mass, we conjecture that the effect of mass should not be ignored when dealing with snake robots, and purely kinematic studies may ignore several of the most important characteristics of a snake robot.

It is the purpose of this line of research to better explain and study a snake robot in a task not related to locomotion, but related to manipulation and dexterous interaction with the robot's environment or external objects. As previously discussed in our thought experiment, understanding the physical limitations of such system is important. It is not only a matter of designing

a controller for choosing the *best* input, but to also understand the overall characteristics of the system. It is intuitively clear, that a snake robot could not manipulate an object that is significantly heavier than the robot itself. Therefore, before understanding manipulation, it is useful to understand the idea of *pushing* the object first. Manipulation would be a natural extension, which relates to the directions the object can be pushed to.

Our system consists of a snake robot in contact with one object to be pushed and the objective is to study the force that can be applied to the object. In addition, the snake robot is contacting the environment with passive wheels. This adds extra friction constraints, something that robotic manipulators usually do not have. It is necessary to study if the extra friction forces have any impact whatsoever in the force exerted onto the object. We conjecture that there should be optimal situations for this interaction, and that the posture (or shape) of the snake robot should have a significant impact. The mass of the snake robot should also play an important role in understanding what objects could be pushed or not, clearly, something that could not be studied with kinematics alone. At this stage, the analysis is limited to snake robots moving in a plane (not necessarily the horizontal plane).

Preliminary findings have been reported in [25, 26] that study the effect of the robot's posture. It has been found that postures where the center of mass (COM) of the robot is closer to the object and along the line of action (the direction of pushing) are optimal and produce the highest object's acceleration. However, no reason on why these configurations are better was given, relying on the interpretation of the results. In [27] the effect of the friction on the slippage of the snake robot was also studied. It is shown that the extra friction does not affect significantly the force exerted on the object, but minimizes the motion of the snake robot. This is important in order to understand when the robot pushes the object, or when the snake robot itself is being pushed away. This separates the problem into two. On one hand, we have the interaction with an object which resembles the problem of robotic manipulation or pushing. On the other hand, the interaction with the environment resembles more the problem of locomotion with legged robots (interaction between the environment and robot to obtain propulsion).

The objective of this paper is to unify previous findings and to extend the analysis under a framework we call Environment-aided Manipulation (EAM). The purpose of EAM is to study the interaction between a snake robot and external bodies (obstacles or the environment itself) in order to accomplish a task that is not related to pure locomotion. In the same way that research regarding robotic manipulators have benefited from understanding its inertial properties (e.g., [28]) we think snake robots could benefit in the same way. Not only snake robots, but in general robots where there is a coupling between locomotion and manipulation fall in the framework of EAM. The primary objective is to show that the posture of the robot plays a significant role in the acceleration of the object. Furthermore, optimal postures are found, regardless of additional friction forces between the snake robot and the environment, The secondary objective is to show an analytical simplified model of a snake robot pushing an object, which corroborates the findings with the full model.

1.2 List of Publications

The following is a list of the publications that lead to the writing of this thesis. The list is in chronological order, either using the date of publication or the date of the conference, where it corresponds.

1. F. Reyes and S. Ma, "On planar grasping with snake robots: Form-closure with enveloping grasps", in *Proc. IEEE Int. Conf. Robotics and Biomimetics (ROBIO 2014)*, 2014, pp. 556–561 (©[2014] IEEE)
2. F. Reyes, W. Tang, and S. Ma, "Using a planar snake robot as a robotic arm taking into account the lack of a fixed base: Feasible region", in *Proc. IEEE Int. Conf. Intelligent Robots and Syst. (IROS 2015)*, 2015, pp. 956–962 (©[2015] IEEE)
3. F. Reyes and S. Ma, "Modeling of snake robots oriented towards grasping and interaction with the environment", in *In: Proc. Int. Conf. Real-time Computing and Robotics (RCAR 2015)*, not published, 2015
4. F. Reyes and S. Ma, "Snake robots in contact with the environment: Influence of the configuration on the applied wrench", in *Proc. IEEE Int. Conf. Intelligent Robots and Syst. (IROS 2016)*, 2016, pp. 3854–3859 (©[2016] IEEE)
5. F. Reyes and S. Ma, "Snake robots in contact with the environment - influence of the friction on the applied wrench", in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS 2017)*, 2017, pp. 5790–5795. DOI: 10.1109/IROS.2017.8206471 (©[2017] IEEE)
6. F. Reyes and S. Ma, "Studying slippage on pushing applications with snake robots", in *In: Proc. Int. Conf. Real-time Computing and Robotics (RCAR 2017)*, 2017 (©[2017] IEEE)
7. F. Reyes, H. Matsumoto, and S. Ma, "Design and implementation of modular and parametric 3d printed snake robot", *The Robotics and Mechatronics Conference (ROBOMECH 2017)*, vol. 2017, 2A2–A11, 2017. DOI: 10.1299/jsmermd.2017.2A2-A11. [Online]. Available: https://www.jstage.jst.go.jp/article/jsmermd/2017/0/2017_2A2-A11/_article/-char/ja
8. F. Reyes and S. Ma, "Studying slippage on pushing applications with snake robots", *Robotics and Biomimetics*, vol. 4, no. 1, p. 9, 2017, ISSN: 2197-3768. DOI: 10.1186/s40638-017-0065-3. [Online]. Available: https://doi.org/10.1186/s40638-017-0065-3 (©[2017] The Authors)

1.3 Outline of the Thesis

The outline of this thesis is as follows. In Chapter 1 the statement of the problem along an exhaustive literature review is presented. Section 1.2 presents a list of the publications that lead to the writing of this thesis.

Chapter 2 presents the concept of EAM. In particular, Section 2.1 presents the definition of EAM used in this thesis, along the mathematical analysis of this problem. Section 2.2 presents the main kinematic model that has been used in this thesis and related published papers. Section 2.3 presents a complete analysis of the mappings involved when a snake robot is in contact with an object. Necessary conditions for form-closure are presented. Section 2.4 presents a set of sufficient conditions to have form-closure using an envelope grasp using a snake robot. Section 2.5 and Section 2.6 move to a full dynamic of the snake robot without and with object(s), respectively. Section 2.7 discusses the projections of the coupled equations of motion onto the constrained and unconstrained subspaces. Section 2.8 introduces the *slippage ratio* relating the motion of the snake robot and object.

Chapter 3 presents a set of results regarding optimal configurations and postures that maximize the acceleration of an object, when a snake robot pushes it. First, Section 3.1 presents a mathematical model that simplifies a snake robot contacting a body, into a two-body multi-rigid-body system. This simplification allows to gather conclusions based on a geometrical analysis. Section 3.1 analyzes with rigorous mathematical modeling the problem and simplifies in order to gain insight. Section 3.2 discusses the scenarios considered in this thesis, which shows that the conclusions obtained can be applied to a wide range of cases. Section 3.3 discusses analytically and then shows with simulations that there is a set of optimal configurations of the snake robot for pushing an object, regardless of the friction with the ground. Section 3.4 extends the analyses to include the motion of the snake robot and object, and shows that there is also a set of configurations that minimize undesired motion. Section 3.5 shows a comparison between the simplified model obtained in Section 3.1 and the full complex model.

Chapter 4 presents experimental results to validate the formulations presented in previous chapters. Specifically, experiments to verify the performance of good and bad postures of the snake robot, as discussed in Section 3.4 are presented. Section 4.1 presents an overview of the prototype of snake robot developed and experimental setup. A more detailed description of the prototype is given in Appendix B. Section 4.2 shows the experimental results.

Chapter 5 provides an extensive discussion regarding the scope of the analysis presented in this thesis and its place compared to other research regarding robotic systems.

Chapter 6 summarizes the most important findings of this thesis and presents some comments regarding the direction that future research may take regarding EAM.

Appendix A presents the minimal mathematical background necessary to understand this document. Specifically, Appendix A.1 presents an overview of theory of *twists* and *wrenches* which are used to build the dynamic model of out system in later sections. Appendix A.2 presents the basic techniques to describe the dynamic model of an object. Appendix A.4 gives a step-by-step guide on how to add the constraints present in the system and couple the models of several subsystems. Appendix C presents the design of the electronic boards and circuits used in the snake robot prototype. Appendix D presents a hard copy of the libraries developed to control the prototype. Appendix E lists online resources with up-to-date information about the electronics and libraries discusses in Appendix C and Appendix D.

Chapter 2

Foundations of EAM

2.1 The Concept of EAM

The concept of **Environmental-aided Manipulation** (EAM) refers to the use of the extra interaction with the environment to accomplish a task related to manipulation. This interaction with the environment can be achieved in several ways, sometimes unintentionally. Originally, we would consider our system (we use the term *system* referring to the mathematical concept of a dynamic system, which may be composed of several subsystems) to be composed of only the snake robot. However the extra interaction with the environment must be accounted for. How does this interaction affect the system?

For example, regarding locomotion, mobile robots obtain propulsive forces by interacting with the ground. Regardless of the physical means to obtain this propulsion (for example, wheels, threads or legs), the principle is the same. Although snake robot's locomotion, inspired in gaits of its biological counterparts seems more complicated, it has been proven that the underlying principle is the same [43, 52], in particular for the most common of gaits: undulatory locomotion (also called slithering). A mechanical system pushes against the ground and is propelled in the opposite direction.

The combination of locomotion and manipulation is a tricky endeavor since it is composed of two complicated tasks. As discussed in Section 1.1, it is common to separate (at least in principle) locomotion and manipulation. For some robotic systems which are composed of distinct sections this is relatively easy. For example, in [1, 2] a robot can stop locomoting and concentrate in manipulating using their robotic arm. The interaction with the environment, and its influence in the manipulation task is therefore minimized or simply ignored.

Snake robots cannot easily avoid this interaction due to the fact that they are always contacting the floor. This extra interaction, which can be modeled as both static or dynamic friction depending on the state of the system, can either help or hinder a snake robot trying to perform a manipulation task. Using this extra friction to help in a manipulation task is a type of EAM. The new system now consists not only of the snake robot, but also includes the interaction with the environment.

Consider the following though experiment (cf. Fig. 2.1).

Listing 2.1: Thought Experiment

```
A snake robot is in contact with an object with the purpose to move
     such object into a desired direction. In contrast, we may be
    interested in stop it from moving given an external undesirable
    force.

The snake robot is in contact with the ground (environment) with
    its belly. It may also be contacting other obstacles, for
    example, a wall.

If the snake robot tries to move the object, what will happen? Is
    the snake robot going to move the object, or is the object so
    heavy that the robot will move instead?
```

These two situations, the robot *moving the object* or the robot *moving around the object*, are normally studied separately. However, under the framework of EAM they are actually proven to be related; they are simply extremes of the same system. Which one is preferable will depend on the task at hand.

If the task is to move the object, then the movement of the robot must be minimized and the situation depicted in Fig. 2.1(b) is desirable. On the other hand, if the task is to locomote while minimizing interaction with the environment and external objects, then the situation depicted in Fig. 2.1(c) is the one desired. Notice that the later is related to the concept of climbing [53, 54, 55, 56].

In this thesis, we make almost not distinction between **grasping** and **manipulation**. The main principle behind them is to have a robotic system impart a force onto an object, and therefore control the object's behavior. **Grasping** deals mainly with the task of constraining an object (static equilibrium), while **manipulation** mainly refers to making an object move in a desired way (dynamic equilibrium). We will make a clear distinction when necessary, but otherwise treat them without distinction.

Since a robot maybe contacting the environment and external objects, it is necessary to be a little more clear on what types of interaction EAM is considering. A few terms used throughout this thesis need further clarification. The framework of EAM uses the next terms:

Definition 1. *Object:* refers to a rigid object(s) to be manipulated. The full inertial parameters and equations (eqns.) of motion are considered along the eqns. of motion of the snake robot. However, there is an exception that will be explained later.
Definition 2. *Obstacle:* refers to any rigid object(s) that the snake robot contacts, but without the purpose to manipulate it. The eqns. of motion of the obstacles must be considered, too. The exception is when the obstacle is fixed to the environment or has infinite mass. In this case we may use the term *static obstacle* or *environment*, as explained in the next definition.

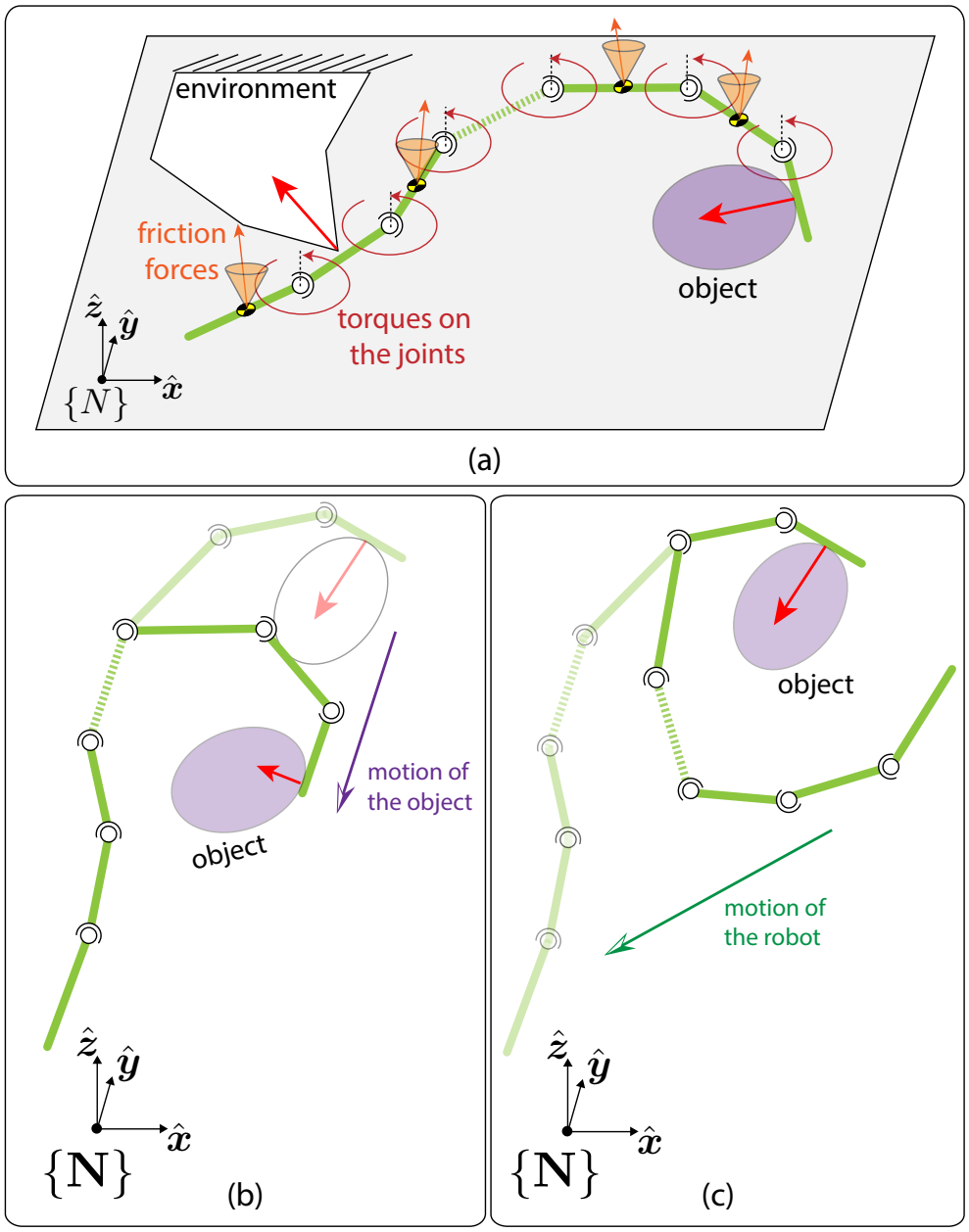


FIGURE 2.1: **Thought Experiment**
General scenario of a snake robot contacting an object. (a) The snake robot contacts an object while it may also be contacting the environment either with its belly (friction) or pushing against a wall, for example. (b) The snake robot may be able to move the object. (c) The object may be very heavy and the snake robot will move around the object

Definition 3. *Environment:* refers to any object external to the snake robot. Unlike an obstacle, the environment is considered to be part of an inertial frame and no eqns. of motion are considered. An example would be a wall or a pole attached to the ground. The environment can be considered an object with infinite mass.

Definition 4. *Constraint forces:* refers to any type of force that imposes a constraint between two objects. The symbol to denote constraint forces is f_c or λ, depending on the space studied (more details in Appendix A.4).

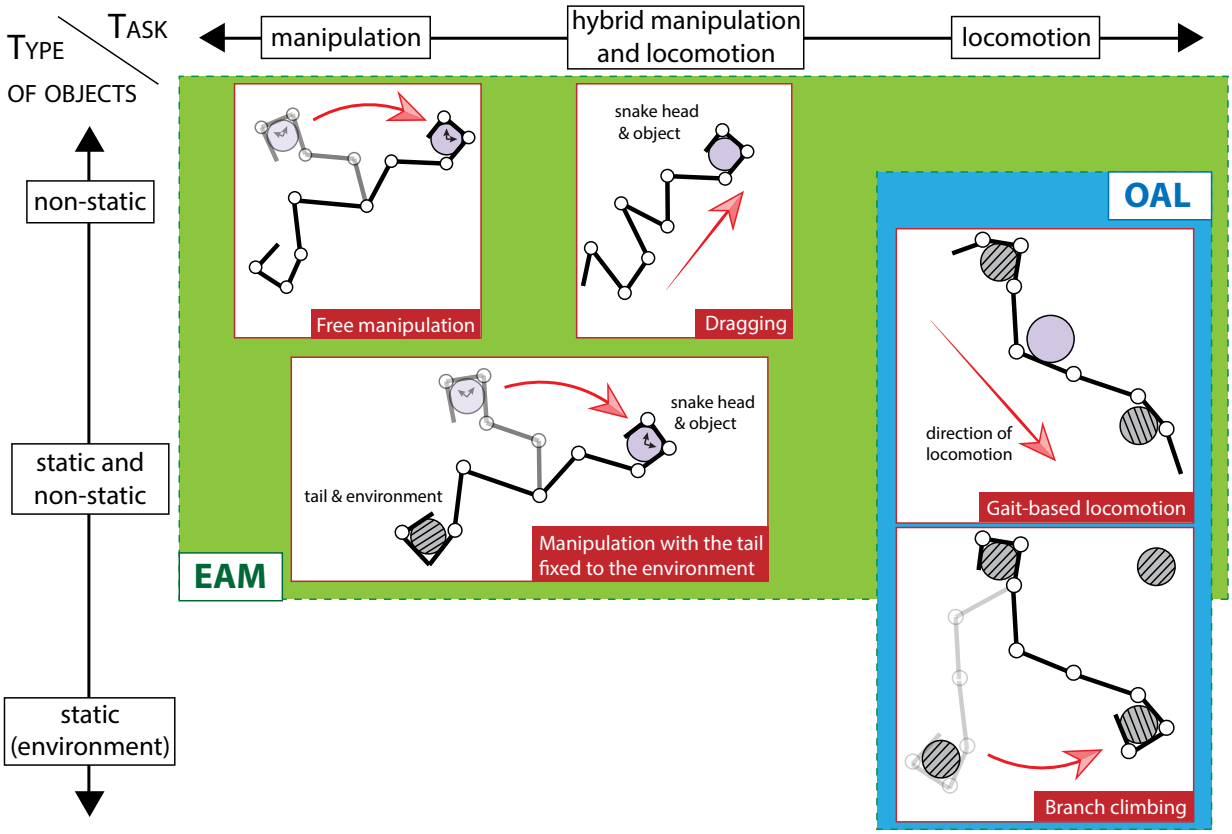


FIGURE 2.2: **EAM Taxonomy**
Comparison of EAM and OAL depending on type of objects and task

Definition 4.1. *Static friction forces:* Special case of constraint forces. In this thesis, a simple model of Coulomb friction is assumed. In other words, friction forces between two objects, in the constrained directions, are limited to their respective *limit surface* [37], which is controlled with a friction coefficient μ_s. The term *static* is dropped since the distinction between static and kinetic friction should be clear from context. The symbol to denote (static) friction forces is $\boldsymbol{f}_{c,f}$ or $\boldsymbol{\lambda}_f$ (more details in Appendix A.4.2).

Definition 4.2. *Non-penetration constraint forces:* Special case of constraint forces. These forces appear due to the fact that rigid bodies cannot penetrate each other. These forces are also commonly called kinematic constraints [35, 37, 39, 40]. In this thesis, these forces are assumed to be unilateral (two objects can push against each other, but not pull) and unbounded. The symbol to denote non-penetration (kinematic) forces is $\boldsymbol{f}_{c,np}$ or $\boldsymbol{\lambda}_{np}$ (more details in Appendix A.4.1).

The whole system then, is composed of the eqns. of motion of the snake robot, object(s), and obstacle(s), along constraint forces between them. We will show that, when an obstacle is heavy enough, it behaves (instantaneously) like if it were attached to the environment and its eqns. of motion can be ignored. Although the environment does not have eqns. of motion, it is still necessary to consider the constraint forces between the snake robot and the environment. For example, static friction between the ground and snake robot can be modeled as a constraint force. Additionally, the constraint that dictates that two objects cannot penetrate each other imposes a set of kinematic constraints, which then give raise to a set of forces.

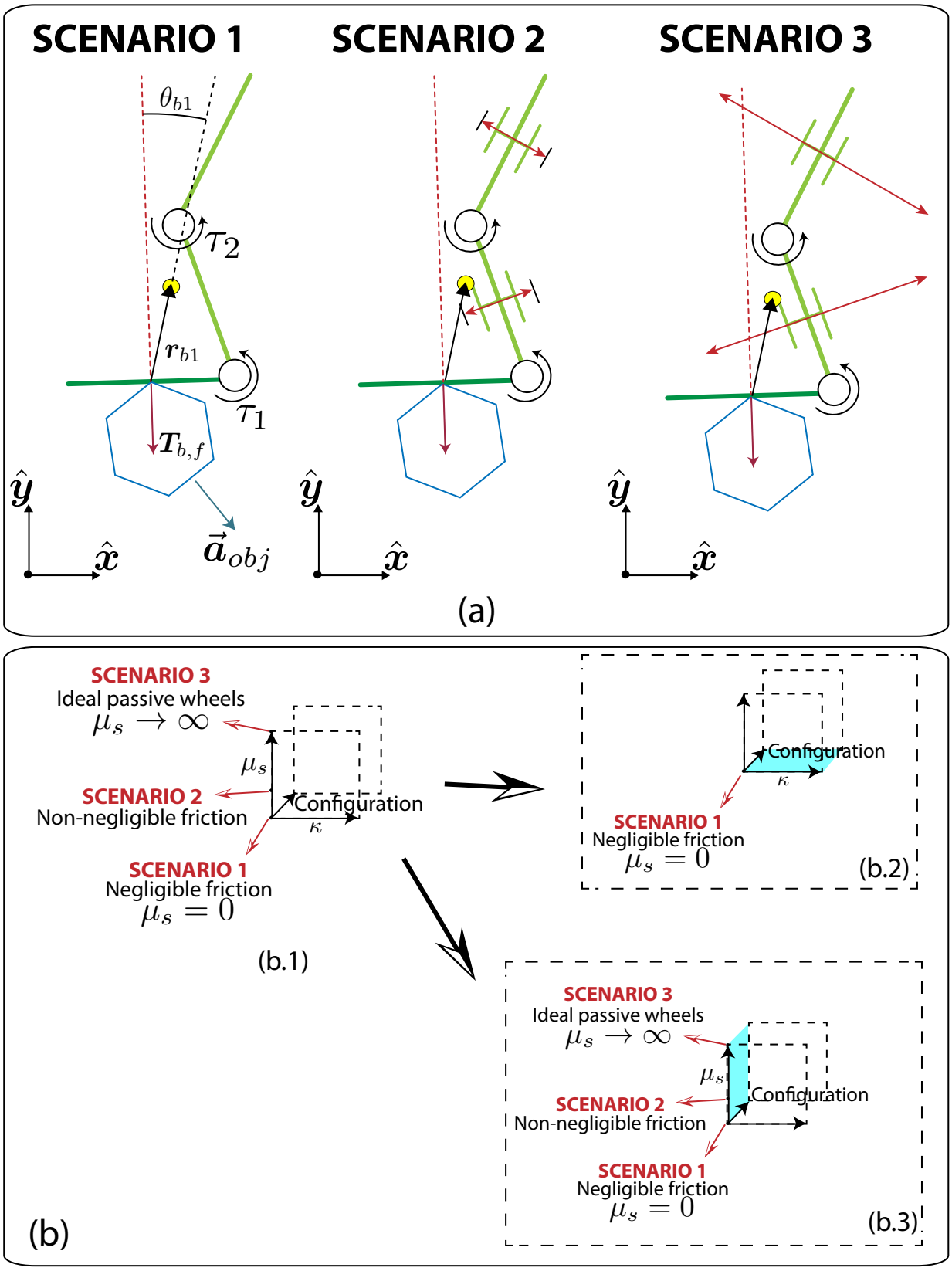


FIGURE 2.3: **Scenarios and Parameters Considered**
(a) All three scenarios considered. From no friction, to ideal unbounded friction
(b) Hypercube of parameters. It encompasses all possible combinations of friction, configuration of the snake robot, and ratio of masses

A special notation that is adopted in this thesis for the constraint forces is the following: The constraint forces will adopt as parameters the systems that are involved in them. For example, if the snake robot is contacting a k-th obstacle, the (non-penetration) constraint forces are written as $\boldsymbol{\lambda}_{np}(\text{snake-robot}, \text{obstacle}_k)$, denoting that the constraint forces couple these two subsystems. Since the environment is not considered to have eqns. of motion, the friction between the snake robot and ground (or any other part obstacle attached to the inertial frame) can be written as $\boldsymbol{\lambda}_f(\text{snake-robot})$.

This allows use to put the concept of EAM in a similar framework to other robotic research. In particular, the concept of Obstacle-aided Locomotion (OAL) [20] is similar to EAM, where the task is locomotion and the robot contacts static objects (the environment). Fig. 2.2 shows how EAM and OAL are related; the main difference is the objective. Consequently, we consider OAL to be directly related to climbing [56]. However, as we will show in Chapter 2, both EAM and OAL are extremes of the same underlying mathematical model, and therefore can be studied on the same mathematical framework. Which approach should be taken is mainly decided by the inertial parameters of the external objects. Even if the robot's objective is to locomote, if the robot is contacting objects that are not static, then the EAM framework presented in this thesis is preferable since it considers the dynamic eqns. of the objects.

There are several parameters and variables that can be studied in such a complex system. As discussed previously, the mass of the several subsystems play a very important role in the behavior of the robot. The ratio between masses (c.f. (2.50)) is extremely important, since it allow us to study any object assuming a general snake robot with unitary mass per link (i.e. each link has 1[kg] of mass).

However, we believe that the most important parameter to study is the friction between the snake robot and the ground. In real applications, it is not possible to know beforehand the friction coefficient between a snake robot and the ground. Even in ideal situations, this problem has been shown to be a stochastic problem, and the problem is less easy to predict the higher the friction coefficient [57]. It is then interesting to consider extreme cases of friction: unbounded and negligible.

Since friction is one of the most important parameters to be studied, it is important to provide a wide range of *scenarios* that consider several different cases of friction. Fig. 2.3(a) provides an overview of the Scenarios considered. They are explained more completely in Section 3.2.

Generally speaking we consider the extremes of negligible friction and ideal unbounded friction, which is the model most commonly used for locomotion with snake robots. Friction is parameterized by its friction coefficient μ_s. Additionally, we consider the ratio between the mass of the snake robot and object denoted by κ, which is described in (2.50). Additionally, we consider the configuration of the snake robot (its joints' angles). These three sets of parameters describe a wide range of systems, and can be imagined as an *hypercube* in the parameter space. Fig. 2.3(b) provides an overview of this hypercube. Fig. 2.3(b.1) shows the an abstraction of the three main sets of parameters to be studied: ratio of masses κ, friction coefficient μ_s and configuration (of

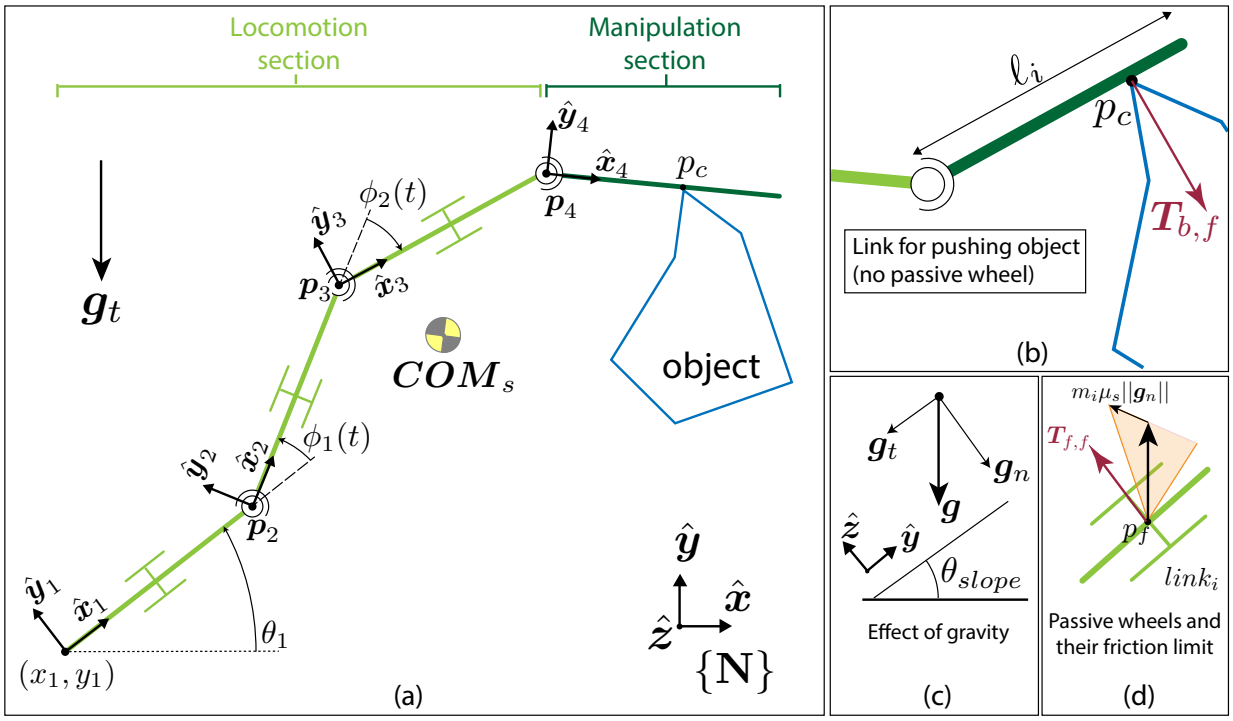


FIGURE 2.4: **Kinematic model**
(a) Kinematic model of the snake robot with respect to an inertial frame {N}. The yellow/black circle denotes the center of mass of the whole snake robot. The snake robot is in contact with an object and the environment. (b) The snake robot exerts a force onto the object. This link does not have a passive wheel. (c) Depending on the slope of the plane, a part of gravity acts on the system. (d) Passive wheels provide friction, depending on the mass of the link and the gravity acting normal to the plane

the snake robot). Fig. 2.3(b.2) shows the plane κ-configuration, which is obtained when assuming no friction with the ground (i.e. $\mu_s = 0$). This plane allows to concentrate on the influence of the configuration and masses. This particular case has been reported in [25]. Fig. 2.3(b.3) shows the plane μ_s-configuration which is obtained when keeping the mass of the object fixed. This plane allows to concentrate on the influence of friction and configurations. This has been reported in [26].

2.2 Kinematic Modeling and Inertial Parameters of the Snake Robot

A snake robot can be modeled as a series of rigid links connected by revolute joints [58, 42]. All joints have their axes parallel to each other, therefore the snake robot is constrained to move on a plane (but unconstrained in any other way). The kinematic model of a snake robot is similar to an open-chain robotic manipulator but with the addition of a floating base (c.f. Fig. 2.4(a)).

The snake robot has a total of $n_s \in \mathbb{N}$ degrees of freedom (DOFs) and its generalized coordinates are encapsulate in the vector $\boldsymbol{q}_s(t) \in \Re^{n_s}$. The first three coordinates account for the non-actuated DOFs of the floating base $\boldsymbol{q}_{na}(t) = [x_1(t), y_1(t), \theta_1(t)]^T \in \Re^{3\times 1}$, where the position of the floating base and its orientation w.r.t. an inertial frame {**N**} are denoted as $[x_1(t), y_1(t)]^T \in \Re^2$ and

$\theta_1(t) \in \Re$, respectively. There are $n_a = n_s - 3$ actuated DOFs (the joints of the snake robot) encapsulated in the vector $\boldsymbol{q}_a(t) = [\phi_1(t), \cdots, \phi_{n_a}(t)]^T \in \Re^{n_a \times 1}$. The vector of generalized coordinates of the snake robot is the concatenation of these last two vectors $\boldsymbol{q}_s(t) = [\boldsymbol{q}_{na}(t)^T \quad \boldsymbol{q}_a(t)^T]^T$, $\boldsymbol{q}_s \in \Re^{n_s \times 1}$.

The snake robot is composed of $n_\ell = n_a + 1$ links each with mass m_i. Then, the total mass of the snake robot is $m_T = \sum_{i=1}^{n_\ell} m_i$. Under the assumption that all links have the same mass m_s, then $m_i = m_s \forall i$ will be used to denote the mass on any link of the snake robot, and the total mass can be simplified as $m_T = n_\ell m_s$.

The set of Cartesian coordinates of the COM of the snake robot is denoted by $\boldsymbol{COM}_s \in \Re^{op}$, where n_{op} denotes the dimensions of the operational space. The coordinates $\boldsymbol{COM}_s$ are referenced w.r.t. the inerial frame {**N**}.

The Jacobian for the i-th link is a mapping from the vector of generalized velocities $\dot{\boldsymbol{q}}_s$ to the twist $\boldsymbol{v}_i \in \Re^3$ of the link and is denoted as $\boldsymbol{J}_i$

$$\boldsymbol{v}_i = \boldsymbol{J}_i \dot{\boldsymbol{q}}_s. \tag{2.1}$$

The twist $\boldsymbol{v}_i$, which contains both linear and angular velocities, does not necessarily represent the velocity of the COM of the i-th link, but rather a quantity that is frame-independent. If the twist needs to be represented w.r.t. a specific frame of reference, transformations like the ones discussed in Appendix A.1 can be used. For example, if the twist is represents the velocity at the origin of the link (cf. Fig. 2.4), and it is wished to know the twist at a frame located at the COM, then the following transformation may be used

$$^{COM}\boldsymbol{v}_i = {}^{COM}\boldsymbol{X}_i \, \boldsymbol{v}_i = {}^{COM}\boldsymbol{X}_i \boldsymbol{J}_i \dot{\boldsymbol{q}}_s, \tag{2.2}$$

where it is assumed that $\boldsymbol{v}_i$ is the twist represented at the origin of the link.

This approach of writing the kinematic and dynamic equations of the system without referencing a specific reference of frame, is discussed in several sources including (but not limited to) [33, 36]. This approach is very powerful since it allows us to derive conclusions based on geometry without looking at the contents of the several matrices and vectors involved. This is an advantage for a multi-dimensional complex system like the one studied in this thesis.

The inertia matrix of the snake robot $\boldsymbol{M}_s \in \Re^{n_s \times n_s}$ can be obtained as

$$\boldsymbol{M}_s = \sum_{i=1}^{i=n_\ell} m_i \boldsymbol{J}_i^T \bar{\boldsymbol{I}}_i \boldsymbol{J}_i, \tag{2.3}$$

where $\bar{\boldsymbol{I}}_i$ denotes the inertia tensor of the i-th link with unitary mass, as discussed in Appendix A. If all the links have the same mass $m_i = m_s \forall i$, then the inertia tensor of the snake robot can be factorized as

$$\boldsymbol{M}_s = m_s \bar{\boldsymbol{M}}_s, \tag{2.4}$$

where the inertia tensor

$$\bar{\boldsymbol{M}}_s := \sum_{i=1}^{i=n_\ell} \boldsymbol{J}_i^T \bar{\boldsymbol{I}}_i \boldsymbol{J}_i \tag{2.5}$$

is defined as the unitary inertia tensor of the snake robot (i.e., the inertia of the snake robot, if all links have unitary mass).

2.3 In-depth Analysis of Grasping with Planar Snake Robots: Form-closure and Subspaces

Most of current literature about snake robots has focused on the study and control of locomotion in planar environments (Please check [58] and the references therein), as discussed in Chapter 1. However, almost no attention has been paid to the study of grasping an object with a snake robot. A main cause of this may be the ambiguity in the definition of *snake robot*. Concepts like *snake-like robot*, *tentacle arm*, *elephant trunk arm*, and *hyper-redundant robot* are used almost interchangeably, causing confusion and problems in the analysis of a specific mechanism.

In this thesis, we consider a snake robot as a robot that mimics the structure of a biological snake, has only revolute joints, does not have an end-effector, and does not have a fixed-base. The lack of fixed-base is critical for distinguishing a snake robot from an hyper-redundant manipulator, since it makes it a mobile robot, and therefore inherently underactuated. In addition, the lack of an end-effector implies that enveloping grasps [59] are of major importance for snake robots. Once these characteristics have been established, it is easier to analyze related research.

In [15], the manipulability of a tentacle arm manipulating an object was studied. In [17] the duality between locomotion and manipulation of a snake robot was considered under the assumption that the snake robot can be treated as a fixed-base manipulator when it is used to manipulate an object. Pipe-climbing locomotion could be tough as a form of grasping, however, in current literature the interaction between the snake robot and pipe (modeled as a rigid-body contact) has not been modeled within the framework of grasping theory [53]. In [29] the requirements for form-closure with a snake robot have been considered. However, only relationships between the size of the object and the links of the robot were considered, without analyzing the kinematic structure of the system.

The purpose of this paper is to analyze the characteristics of a snake robot grasping an object. In particular, we analyze the mappings between subspaces created by the introduction of the kinematic constraints introduced by the contact between the snake robot and object. Several papers have studied the general characteristics of these subspaces, however, in a general way and not considering a mobile robot [39, 40, 60]. Therefore, a more detailed analysis is necessary to understand the properties of a snake robot. Necessary conditions for grasps with form-closure are also studied.

The following assumptions are considered in this analysis:

- Assumption 1: We consider rigid-body contacts. This means compliance is not considered. It has been pointed out [37, 39] that this may lead to hyper-static grasps. However, we give conditions so that this does not happen.
- Assumption 2: Only frictionless point contacts are considered.
- Assumption 3: The snake robot and object are restricted to move only in the horizontal plane.
- Assumption 4: The object has a circular cross-section. This allows to study the subspaces and conditions for form-closure in a more general way.
- Assumption 5: For simplicity, it is assumed that the center of mass (COM) of the object is at the centroid. Further comments can be found in Section 2.3.2.
- Assumption 6: The width of the links of the snake robot is ignored. This facilitates the analysis, and is justified since the width of snake robots is much smaller than their length.

Vectors and matrices will be denoted with bold letters and unitary vectors with a caret, like $\hat{\boldsymbol{x}}$. A right superscript on a quantity denotes the frame on which such quantity is expressed. Unless otherwise stated, quantities are expressed in the inertial frame $\{\mathbf{N}\}$. All frames are assumed to be right-handed.

2.3.1 Kinematic Modeling Notes

This section provides a more in-depth kinematic analysis of the snake robot targeted towards the analysis of grasping. The kinematic model of a snake robot is similar to an open-chain robotic manipulator. The reference frames can be assigned in any way, like in the Denavit-Hartenberg convention [38]. The lack of a fixed-base introduces three extra degrees of freedom (DOFs) that are not actuated. For details please see Fig. 2.4.

The pose (position and orientation) of the object can be described by the position of the origin $\boldsymbol{p}_{obj}(t) \in \Re^{2\times 1}$ and orientation of a frame $\{\mathbf{O}\}$ attached to the center of mass (COM) of the object. Both quantities are encapsulated in the vector $\boldsymbol{q}_{obj}(t) = \left[\boldsymbol{p}_{obj}(t)^T, \theta_{obj}(t)\right]^T \in \Re^{3\times 1}$.

The dimension of the operational space of the object is denoted as $n_\nu \in \mathbb{Z}_+$ The twist of the object $\boldsymbol{\nu}_{obj}(t) \in \Re^{6\times 1}$ contains the linear $\boldsymbol{v}(t) \in \Re^{3\times 1}$ and angular velocity $\boldsymbol{\omega}(t) \in \Re^{3\times 1}$ of the object, i.e., $\boldsymbol{\nu}_{obj}(t) = [\boldsymbol{v}(t), \boldsymbol{\omega}(t)]^T$. Special considerations for the planar case will be discussed as necessary.

Constraints imposed by the contact between rigid bodies

We assume there are $n_c \in \mathbb{Z}_+$ contact points. A reference frame $\{\mathbf{C}\}_k$ is set at the *k-th* contact point (cf. Fig. 2.5). The reference frame $\{\mathbf{C}\}_k$ is related to frame $\{\mathbf{N}\}$ by the rotation matrix

$$\boldsymbol{R}c_k = \begin{bmatrix} \hat{\boldsymbol{x}}_{ck} & \hat{\boldsymbol{y}}_{ck} & \hat{\boldsymbol{z}}_{ck} \end{bmatrix} \tag{2.6}$$

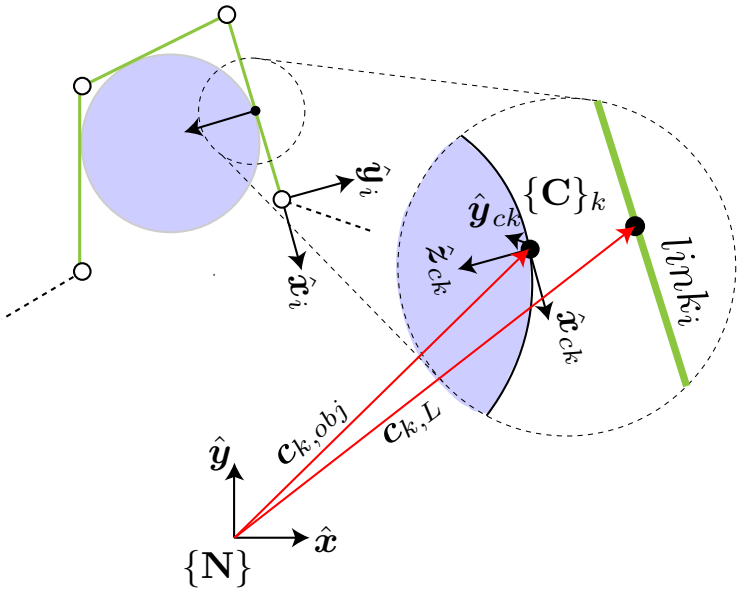


FIGURE 2.5: **Contact between snake robot and object**
Exploited view of the *k-th* contact c_k between the object and $link_i$ of the snake robot.

where the unitary vectors $\hat{x}_{ck}, \hat{y}_{ck}, \hat{z}_{ck} \in \Re^{3\times1}$ form a basis for the frame $\{\mathbf{C}\}_k$. The $\hat{z}_{ck}$ axis is normal to the tangent plane at the contact and directed inward to the object. An augmented matrix $\bar{R}c_k \in \Re^{6\times6}$ that will be useful later, can be constructed as:

$$\bar{R}c_k = \begin{bmatrix} Rc_k & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & Rc_k \end{bmatrix}. \tag{2.7}$$

The three main models of contact between rigid-bodies are: frictionless point contact, hard finger, and soft finger [35, 37]. These models express the motions that are constrained by the contact between the snake robot and object. This paper focuses on frictionless point contacts since form-closure does not consider friction. This type of contact can only constrain motion in the normal direction of the plane located at the contact, that is, in the direction of the $\hat{z}_{ck}$ axis.

A selection matrix $H \in \Re^{1\times6}$ can be designed to select the constrained directions of motion:

$$H = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{2.8}$$

Given the *k-th* contact point between the object and $link_i$, two points are of interest; the point $c_{k,L} \in \Re^{3\times1}$ attached to $link_i$, and the point $c_{k,obj} \in \Re^{3\times1}$ attached to the object, both coincident to the origin of the frame $\{\mathbf{C}\}_k$. The twists of these contact points are denoted as $\nu_{k,L} \in \Re^{6\times1}$ and $\nu_{k,obj} \in \Re^{6\times1}$, respectively.

The interaction between the snake robot and the object can be described by what is known as the Grasp Matrix G and the Hand Jacobian J_H. Even though a snake robot is not a robotic hand, this term will be used to be consistent with modern literature regarding grasping.

Hand Jacobian

The mapping between the joint velocities $\dot{\boldsymbol{q}}(t)$ and the twist $\boldsymbol{\nu}_{k,L}$ of the contact point on $link_i$ (expressed in the frame $\{\mathbf{C}\}_k$) can be obtained as:

$$\boldsymbol{\nu}_{k,L}^{k} = \bar{\boldsymbol{R}} c_k^T \boldsymbol{Z}_i \dot{\boldsymbol{q}}. \tag{2.9}$$

The matrix $\boldsymbol{Z}_i \in \Re^{6\times n}$ is defined as:

$$\boldsymbol{Z}_i(\boldsymbol{q}(t)) = \begin{bmatrix} \boldsymbol{d}_{i,1} & \cdots & \boldsymbol{d}_{i,n} \\ \boldsymbol{l}_{i,1} & \cdots & \boldsymbol{l}_{i,n} \end{bmatrix} \tag{2.10}$$

where $\boldsymbol{d}_{i,j}, \boldsymbol{l}_{i,j} \in \Re^{3\times 1}$ contain the Plücker coordinates of the axes of the joints [37]. For the specific case of a planar snake robot, and having assigned the reference frames as in Fig. 2.4, these vectors can be defined as:

$$\boldsymbol{d}_{i,j} = \begin{cases} \hat{\boldsymbol{z}}_0 & \text{for } j = 1, \forall i \\ \hat{\boldsymbol{z}}_1 & \text{for } j = 2, \forall i \\ \boldsymbol{S}(\boldsymbol{c}_{k,L} - \boldsymbol{p}_{j-1})^T \hat{\boldsymbol{z}}_{j-1} & \text{for } j = 3, ..., i \\ \mathbf{0}_{3\times 1} & \text{for } j = i+1, ..., n \end{cases} \tag{2.11}$$

$$\boldsymbol{l}_{i,j} = \begin{cases} \mathbf{0}_{3\times 1} & \text{for } j = 1, \forall i \\ \mathbf{0}_{3\times 1} & \text{for } j = 2, \forall i \\ \hat{\boldsymbol{z}}_{j-1} & \text{for } j = 3, ..., i \\ \mathbf{0}_{3\times 1} & \text{for } j = i+1, ..., n \end{cases} \tag{2.12}$$

where $\boldsymbol{S}(\bullet)$ denotes the cross-product matrix (also known as skew-symmetric operator) and $\hat{\boldsymbol{z}}_j$ is the z-axis of the *j-th* frame. In other words, the *j-th* column of $\boldsymbol{Z}_i$ would be $\mathbf{0}$ if $\dot{\boldsymbol{q}}_j$ does not contribute to the velocity of the contact point. The vectors $\boldsymbol{d}_{i,j}, \boldsymbol{l}_{i,j}$ have some special features:

- $\boldsymbol{d}_{i,j}$ for $j \geqslant 3$ either lies on the plane or is $\mathbf{0}_{3\times 1}$.
- $\boldsymbol{l}_{i,j}$ for $j \geqslant 3$ is either perpendicular to the plane or is $\mathbf{0}_{3\times 1}$.
- $\hat{\boldsymbol{x}}_{ck}$ and $\hat{\boldsymbol{z}}_{ck}$ lie on the plane $\forall k$.
- $\hat{\boldsymbol{y}}_{ck}$ is perpendicular to the plane $\forall k$.

By expanding the term $\bar{\boldsymbol{R}}c_k^T\boldsymbol{Z}_i$, and taking into account (2.7), (2.11), (2.12), and the previous considerations, we can arrive at the next expression:

$$\bar{\boldsymbol{R}}_{ck}^T\boldsymbol{Z}_i = \begin{bmatrix} \hat{\boldsymbol{x}}_{ck}^T\hat{\boldsymbol{z}}_0 & \hat{\boldsymbol{x}}_{ck}^T\hat{\boldsymbol{z}}_1 & \hat{\boldsymbol{x}}_{ck}^T\boldsymbol{d}_{i,3} & \cdots & \hat{\boldsymbol{x}}_{ck}^T\boldsymbol{d}_{i,n} \\ 0 & 0 & 0 & \cdots & 0 \\ \hat{\boldsymbol{z}}_{ck}^T\hat{\boldsymbol{z}}_0 & \hat{\boldsymbol{z}}_{ck}^T\hat{\boldsymbol{z}}_1 & \hat{\boldsymbol{z}}_{ck}^T\boldsymbol{d}_{i,3} & \cdots & \hat{\boldsymbol{z}}_{ck}^T\boldsymbol{d}_{i,n} \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \hat{\boldsymbol{y}}_{ck}^T\boldsymbol{l}_{i,3} & \cdots & \hat{\boldsymbol{y}}_{ck}^T\boldsymbol{l}_{i,n} \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}, \tag{2.13}$$

where the nature of the planar snake robot becomes clear. The 1^{st} and 3^{rd} rows correspond to the linear velocities of the contact points in the direction of the $\hat{\boldsymbol{x}}_{ck}$ and $\hat{\boldsymbol{z}}_{ck}$ axes, respectively. The 5^{th} row correspond to the angular velocity around the $\hat{\boldsymbol{y}}_{ck}$ axes. All quantities expressed in the local frame $\{\mathbf{C}\}_k$. Now, premultiply by the selection matrix $\boldsymbol{H}$, in order to take into account the rigid-body model constraints (this simply selects the 3^{rd} row of (2.13)):

$$\boldsymbol{J}_i = \begin{bmatrix} \hat{\boldsymbol{z}}_{ck}^T\hat{\boldsymbol{z}}_0 & \hat{\boldsymbol{z}}_{ck}^T\hat{\boldsymbol{z}}_1 & \hat{\boldsymbol{z}}_{ck}^T\boldsymbol{d}_{i,3} & \cdots & \hat{\boldsymbol{z}}_{ck}^T\boldsymbol{d}_{i,n} \end{bmatrix}. \tag{2.14}$$

Finally, by stacking the Jacobians of all contact points, the Hand Jacobian $\boldsymbol{J}_H \in \Re^{n_c \times n}$ can be obtained:

$$\boldsymbol{J}_H = \begin{bmatrix} \boldsymbol{H}\bar{\boldsymbol{R}}c_1^T\boldsymbol{Z}_* \\ \vdots \\ \boldsymbol{H}\bar{\boldsymbol{R}}c_{n_c}^T\boldsymbol{Z}_* \end{bmatrix} \tag{2.15}$$

where the index $*$ in $\boldsymbol{Z}_*$ indicates the contacting $link_*$ at the *k-th* contact.

In summary, the Hand Jacobian $\boldsymbol{J}_H$ is a linear mapping between the subspace of generalized coordinates velocities $\dot{\boldsymbol{q}}$ and the elements of the twist of the contact point $\boldsymbol{\nu}_{k,L}^k$ that are constrained by the rigid-body contact ($\boldsymbol{J}_H : \Re^n \rightarrow \Re^{n_c}$).

Grasp Matrix

In a similar fashion as in the previous section, a relationship between the twist of the object $\boldsymbol{\nu}_{obj}$ and the twists of the contact points $\boldsymbol{\nu}_{k,obj}^k$ can be found as:

$$\boldsymbol{\nu}_{ck,obj}^k = \bar{\boldsymbol{R}}c_k^T\boldsymbol{D}_k\boldsymbol{\nu}_{obj} \tag{2.16}$$

where

$$\boldsymbol{D}_k = \begin{bmatrix} \boldsymbol{I}_{3\times3} & -\boldsymbol{S}(\boldsymbol{c}_{k,obj} - \boldsymbol{p}_{obj}) \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} \end{bmatrix} \in \Re^{6\times6} \tag{2.17}$$

It is worth expanding the term $\bar{\boldsymbol{R}}c_k^T\boldsymbol{D}_k$:

$$\bar{\boldsymbol{R}}c_k^T\boldsymbol{D}_k = \begin{bmatrix} \hat{\boldsymbol{x}}_{ck}^T & (\boldsymbol{S}(\boldsymbol{c}_{k,obj} - \boldsymbol{p}_{obj})\hat{\boldsymbol{x}}_{ck})^T \\ \hat{\boldsymbol{y}}_{ck}^T & (\boldsymbol{S}(\boldsymbol{c}_{k,obj} - \boldsymbol{p}_{obj})\hat{\boldsymbol{y}}_{ck})^T \\ \hat{\boldsymbol{z}}_{ck}^T & (\boldsymbol{S}(\boldsymbol{c}_{k,obj} - \boldsymbol{p}_{obj})\hat{\boldsymbol{z}}_{ck})^T \\ \boldsymbol{0}_{1\times 3} & \hat{\boldsymbol{x}}_{ck}^T \\ \boldsymbol{0}_{1\times 3} & \hat{\boldsymbol{y}}_{ck}^T \\ \boldsymbol{0}_{1\times 3} & \hat{\boldsymbol{z}}_{ck}^T \end{bmatrix} \tag{2.18}$$

Now, premultiply the previous expression by the selection matrix $\boldsymbol{H}$ in order to get the matrix $\boldsymbol{G}_k^T \in \Re^{1\times 6}$:

$$\boldsymbol{G}_k^T = \begin{bmatrix} \hat{\boldsymbol{z}}_{ck}^T & (\boldsymbol{S}(\boldsymbol{c}_{k,obj} - \boldsymbol{p}_{obj})\hat{\boldsymbol{z}}_{ck})^T \end{bmatrix}. \tag{2.19}$$

The transpose of the Grasp Matrix is obtained by concatenating these expressions for the n_c contacts as

$$\boldsymbol{G}^T = \begin{bmatrix} \boldsymbol{G}_1^T \\ \vdots \\ \boldsymbol{G}_{n_c}^T \end{bmatrix} \tag{2.20}$$

where $\boldsymbol{G}^T \in \Re^{n_c\times 6}$. The structure of the matrix $\boldsymbol{G}^T$ has some considerations as well:

- Since axis $\hat{\boldsymbol{z}}_{ck}$ lie on the plane $\forall k$, its third component is 0.
- the vector $(\boldsymbol{S}(\boldsymbol{c}_{k,obj} - \boldsymbol{p}_{obj})\hat{\boldsymbol{z}}_{ck})^T$ is $\boldsymbol{0}_{1\times 3}$ $\forall k$ since these vectors are parallel. Notice that this is a direct result of having considered the COM of the object at the centroid.

Taking into account these considerations, the matrix $\boldsymbol{G}^T$ has the following general structure:

$$\boldsymbol{G}^T = \begin{bmatrix} * & * & 0 & 0 & 0 & 0 \\ & & \vdots & & & \\ * & * & 0 & 0 & 0 & 0 \end{bmatrix}$$

where $*$ indicates possibly non-zero elements. This again, shows the planar nature of the system, since some elements of the twist of the object $\boldsymbol{\nu}_{obj}$ have no contribution to the twist of the contact point $\boldsymbol{\nu}_{k,obj}^k$.

The matrix (2.20) is intended for spatial cases ($n_\nu = 6$). For the remainder of this paper it will be assumed that $n_\nu = 3$, that is, the operational space is the plane (the 1^{st}, 2^{nd} and 6^{th} columns of (2.20) are kept). Then, $\boldsymbol{G}^T$ would look like:

$$\boldsymbol{G}^T = \begin{bmatrix} * & * & 0 \\ & \vdots & \\ * & * & 0 \end{bmatrix} \tag{2.21}$$

To avoid abuse of indexes no distinction will be made when using the Grasp Matrix assuming its dimensions are clear from context.

In summary, the transpose of the Grasp Matrix $\boldsymbol{G}^T$ is a linear mapping between the subspace twist of the object $\boldsymbol{\nu}_{obj}$ and the elements of the twist of the contact point $\boldsymbol{\nu}^k_{k,obj}$ that are constrained by the rigid-body contact ($\boldsymbol{G}^T : \Re^6 \to \Re^{n_c}$).

Kinematic Constraints

The general form of constraints has been discussed in Appendix A.4. Here, we present the specific form they take when dealing with a snake robot contacting an object. This formulation is compatible with research regarding grasping with robotic hands [37] and other systems [39, 40, 35].

The constraints imposed by contact between rigid bodies can be expressed as:

$$\boldsymbol{A} \begin{bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{\nu}_{obj} \end{bmatrix} \geqslant \boldsymbol{0} \tag{2.22}$$

where the constraint matrix $\boldsymbol{A} \in \Re^{n_c \times (n+n_v)}$ is given by:

$$\boldsymbol{A}(\boldsymbol{q}, \boldsymbol{q}_{obj}) = \begin{bmatrix} -\boldsymbol{J}_H & \boldsymbol{G}^T \end{bmatrix}. \tag{2.23}$$

Equation (2.22) simply states that the object cannot penetrate the snake robot. These constraints are holonomic, since they could have been expressed as a function of the configuration of the system ($\boldsymbol{q}(t)$ and $\boldsymbol{q}_{obj}(t)$), instead of the velocities ($\dot{\boldsymbol{q}}(t)$ and $\boldsymbol{\nu}_{obj}(t)$). However, it is more convenient to treat them at the velocity level, since the mappings are linear [33, 51, 61, 62]. The set of constraints (2.22) are unilateral, however, when considering grasps with closure it is acceptable to consider bilateral constraints, delegating the task of calculating and maintaining proper contact forces to a controller [40]. Since the constraints state that $\boldsymbol{\nu}^k_{k,L}$ and $\boldsymbol{\nu}^k_{k,obj}$ must be the same in the constrained directions (i.e., in the direction of $\hat{\boldsymbol{z}}_{ck}$), for simplicity we will call both these motions *contact twists* $\boldsymbol{\nu}_{cc} \in \Re^{n_c \times 1}$.

Basic properties of a grasp

There are several concepts that are relevant when studying constraints (2.22) [37, 35]. Given a set of contacts, one of the most important properties of a grasp is the concept of *closure*. There are several definitions of *form-closure* and *force-closure* in literature, sometimes even contradictory. In this paper, we adopt the most modern definitions:

- Form-closure: The object is said to be grasped with form-closure, if any motion would violate the constraints (2.22). In other words,the object is immobilized.
- Force-closure: If the mechanism used to grasp the object can exert an arbitrary wrench unto the object, then the grasp has force-closure.

In particular, it is worth mentioning that analysis of form-closure only requires the analysis of $\boldsymbol{G}$. That is, we can analyze the grasp while considering the contact points as static [40]. On the other hand, for the analysis of force-closure, the structure of the snake robot has to be considered along an analysis of the eqns. of motion (sometimes, ignoring inertial terms).

Aside from the concepts of closure, the analysis of the mappings $\boldsymbol{J}_H$, $\boldsymbol{G}^T$, $\boldsymbol{J}_H^T$, and $\boldsymbol{G}$ can give useful insights into the properties of the system. The next classifications of grasps are useful [37]:

- *Redundant*: If the nullspace of $\boldsymbol{J}_H$, defined as

$$\mathcal{N}(\boldsymbol{J}_H) = \{\dot{\boldsymbol{q}} | \boldsymbol{J}_H \dot{\boldsymbol{q}} = \boldsymbol{0}\}, \tag{2.24}$$

  is nontrivial (i.e., $\mathcal{N}(\boldsymbol{J}_H) \neq \boldsymbol{0}$), then the grasp is said to be *redundant*. Any $\dot{\boldsymbol{q}}$ in $\mathcal{N}(\boldsymbol{J}_H)$ does not produce motions of the contact points $\boldsymbol{\nu}_{k,L}^k$ in the constrained.

- *Indeterminate*: If the nullspace of $\boldsymbol{G}^T$, defined as

$$\mathcal{N}(\boldsymbol{G}^T) = \{\boldsymbol{\nu}_{obj} | \boldsymbol{G}^T \boldsymbol{\nu}_{obj} = \boldsymbol{0}\}, \tag{2.25}$$

  is nontrivial (i.e., $\mathcal{N}(\boldsymbol{G}^T) \neq \boldsymbol{0}$), then the grasp is said to be *indeterminate*. Object twists $\boldsymbol{\nu}_{obj}$ in $\mathcal{N}(\boldsymbol{G}^T)$ are motions of the object that do not violate the constraints 2.22.

- *Defective*: If the nullspace of $\boldsymbol{J}_H^T$, defined as

$$\mathcal{N}(\boldsymbol{J}_H^T) = \{\boldsymbol{\lambda} | \boldsymbol{J}_H^T \boldsymbol{\lambda} = \boldsymbol{0}\}, \tag{2.26}$$

  is nontrivial (i.e., $\mathcal{N}(\boldsymbol{J}_H^T) \neq \boldsymbol{0}$), then the grasp is said to be *defective*. Contact forces $\boldsymbol{\lambda}$ in $\mathcal{N}(\boldsymbol{J}_H^T)$ do not produce any effect in the dynamics of the snake robot described by (2.43).

- *Graspable*: If the nullspace of $\boldsymbol{G}$, defined as

$$\mathcal{N}(\boldsymbol{G}) = \{\boldsymbol{\lambda} | \boldsymbol{G}\boldsymbol{\lambda} = \boldsymbol{0}\} \tag{2.27}$$

  is nontrivial (i.e., $\mathcal{N}(\boldsymbol{G}) \neq \boldsymbol{0}$), then the grasp is said to be *graspable*. Contact forces $\boldsymbol{\lambda}$ in $\mathcal{N}(\boldsymbol{G})$ do not produce any effect in the dynamics of the object described by (A.1). These forces are called *internal* forces (or sometimes *squeezing* forces) since they do not contribute to the acceleration of the object, however, they are the very forces that help to grasp the object.

- *Hyper-static*: If some contact forces belong to both $\mathcal{N}(\boldsymbol{J}_H^T)$ and $\mathcal{N}(\boldsymbol{G})$ then the grasp is said to be *hyper-static*. In other words, $\mathcal{N}(\boldsymbol{J}_H^T) \bigcap \mathcal{N}(\boldsymbol{G}) \neq \boldsymbol{0}$. In hyper-static grasps the dynamics given by (2.43), (A.1), and constraints (2.22) do not fully describe the system and compliance has to be introduced into the model [39].

2.3.2 Analysis of the object's mappings

Before stating our objectives, it is necessary to recall some basic concepts of linear algebra. Given any linear mapping $\boldsymbol{A} \in \Re^{o \times p}$ and its rank $(\text{rank}(\boldsymbol{A}) \leqslant \min(o, p))$ the following subspaces are important. Its column space (also called range or image) is denoted as $\mathcal{R}(\boldsymbol{A}) \in \Re^{o}$ and its row space as $\mathcal{R}(\boldsymbol{A}^T) \in \Re^{p}$. Also, recall that $\text{rank}(\boldsymbol{A}) = \text{rank}(\boldsymbol{A}^T)$.

The dimensions of all subspaces of $\boldsymbol{G}^T \in \Re^{n_c \times 3}$ and $\boldsymbol{G} \in \Re^{3 \times n_c}$ can be summarized as:

$$\boldsymbol{G}^T, \boldsymbol{G} \begin{cases} \dim(\mathcal{R}(\boldsymbol{G}^T)) = \text{rank}(\boldsymbol{G}^T) \\ \dim(\mathcal{N}(\boldsymbol{G}^T)) = 3 - \text{rank}(\boldsymbol{G}^T) \\ \dim(\mathcal{R}(\boldsymbol{G})) = \text{rank}(\boldsymbol{G}^T) \\ \dim(\mathcal{N}(\boldsymbol{G})) = n_c - \text{rank}(\boldsymbol{G}^T) \end{cases} \tag{2.28}$$

where $\mathcal{R}(\bullet)$ denotes the column space (also called image or range). The next properties for the grasp are desired:

Objective 1

Partially indeterminate: Since the objective is to constrain the object, it is desired that no motions of the object are possible. This could be achieved if $\dim(\mathcal{N}(\boldsymbol{G}^T)) = 0$ This is not possible for a circular object since an exceptional surface cannot be constrained with frictionless contacts (p. 230 of [35]), however, its translation can still be constrained. In other words, it cannot be avoided that $\dim(\mathcal{N}(\boldsymbol{G}^T)) = 1$. This means that $\text{rank}(\boldsymbol{G}^T) = 3 - \dim(\mathcal{N}(\boldsymbol{G}^T)) = 2$. Actually, this can be confirmed by looking at (2.21), which tells us that it could never have full column rank. Notice that, if the COM were not in the centroid, a couple could be exerted unto the object, and the grasp could be non-indeterminate.

Objective 2

Graspable: Form-closure requires the existence of internal forces. For this to be true, it is necessary that $\mathcal{N}(\boldsymbol{G}) \neq \boldsymbol{0}$. This can be accomplished if $n_c > \text{rank}(\boldsymbol{G}^T)$. Notice this is a necessary, but not a sufficient condition for form-closure.

The involved subspaces are shown in Fig. 2.6. The condition $\text{rank}(\boldsymbol{G}^T) = 2$ combined with $n_c > \text{rank}(\boldsymbol{G}^T)$ implies that at least three frictionless contact points are necessary in order to constrain the translation (but not the rotation) of the object. Under this condition, the grasp could have what is called *partial* form-closure. However, for simplicity we will still refer to it simply as form-closure, with the understanding that only the translation of the object is constrained.

Although this shows the basic properties that the mappings must have in order to have form-closure, more conditions are needed since the internal forces can only be positive. Several points of view exist to check this, and a more rigorous proof of form-closure can be found in the references [37, 63]. As a simple test, if the $\hat{\boldsymbol{z}}_{ck}$ axes positively span the space of object wrenches $\boldsymbol{g}$, then there is form-closure. In the examples presented in 2.3.4 this concepts will be clearly shown.

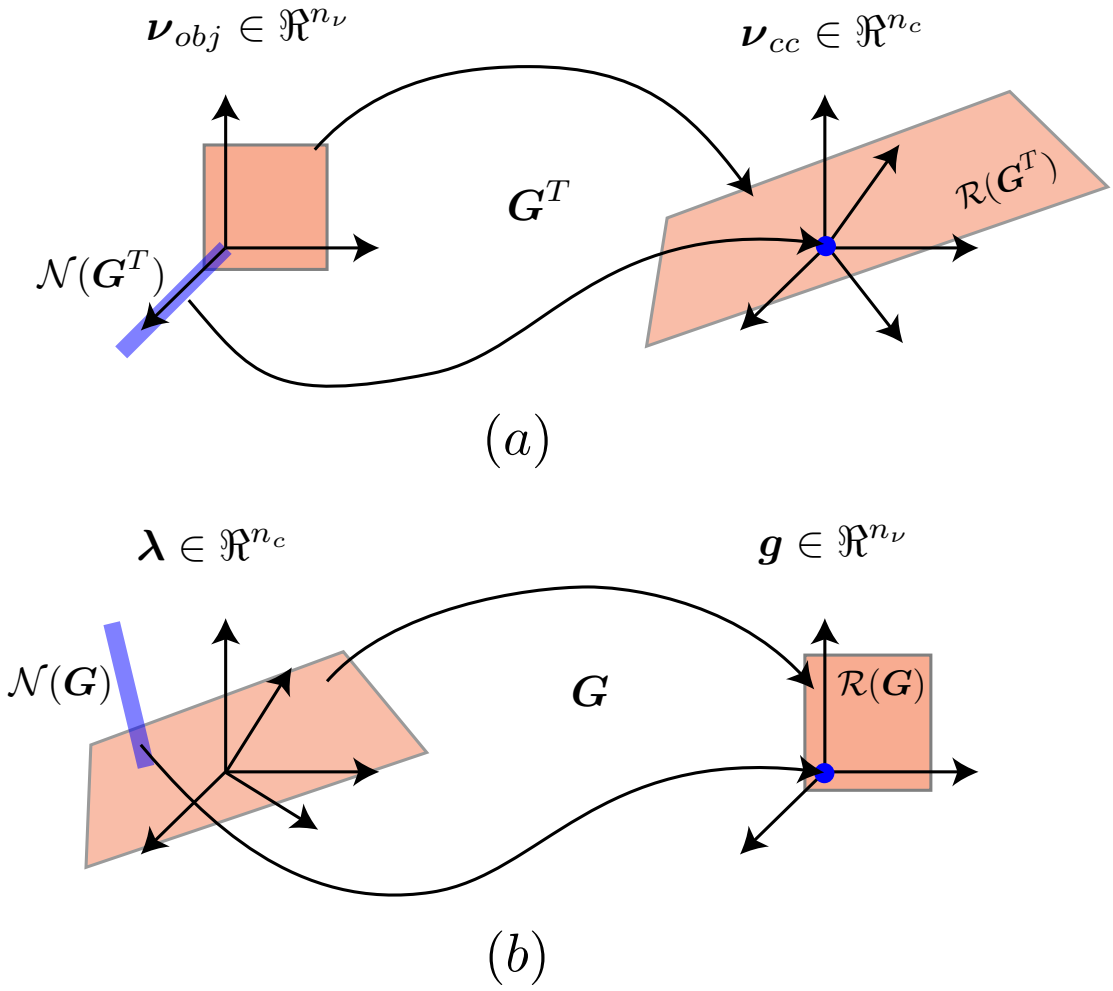


FIGURE 2.6: **Subspaces of G^T and G**
(a) The mapping $\boldsymbol{G}^T$ and its subspaces. Notice that $\mathcal{N}(\boldsymbol{G}^T)$ is nontrivial and the subspace of contact twists $\boldsymbol{\nu}_{cc}$ is not fully accessible.(b) The mapping $\boldsymbol{G}$ and its subspaces. Notice that $\mathcal{N}(\boldsymbol{G})$ is nontrivial. The subspace of wrenches applied to the object $\boldsymbol{g}$ is not fully accessible (a torque cannot be exerted onto the object).

Also, it can be seen that the conditions for form-closure do not consider the structure of the snake robot. In the next section, we analyze if a planar snake robot could generate these forces contact forces.

2.3.3 Analysis of the snake robot's mappings

The dimensions of all subspaces of $\boldsymbol{J}_H \in \Re^{n_c \times n}$ and $\boldsymbol{J}_H^T \in \Re^{n \times n_c}$ can be summarized as:

$$\boldsymbol{J}_H, \boldsymbol{J}_H^T \begin{cases} \dim(\mathcal{R}(\boldsymbol{J}_H)) = \text{rank}(\boldsymbol{J}_H) \\ \dim(\mathcal{N}(\boldsymbol{J}_H)) = n - \text{rank}(\boldsymbol{J}_H) \\ \dim(\mathcal{R}(\boldsymbol{J}_H^T)) = \text{rank}(\boldsymbol{J}_H) \\ \dim(\mathcal{N}(\boldsymbol{J}_H^T)) = n_c - \text{rank}(\boldsymbol{J}_H) \end{cases} \tag{2.29}$$

The next desired properties for the grasp can be analyzed:

Objective 3

Redundant: Redundancy is desired, since it would allow the snake robot to have extra DOFs while maintaining contact with the object. In other words, it is desired that $\mathcal{N}(\boldsymbol{J}_H) \neq \mathbf{0}$. For that, it is necessary that $n > \text{rank}(\boldsymbol{J}_H)$

Objective 4

Non-defective: In order to avoid hyper-static grasps and in general contact forces $\boldsymbol{\lambda}$ that could not be generated by the snake robot, it is desirable that $\mathcal{N}(\boldsymbol{J}_H^T) = \mathbf{0}$. In other words, it is required that $\text{rank}(\boldsymbol{J}_H) = n_c$

Combining both objectives, it is clear that necessary conditions are that $\boldsymbol{J}_H$ is full row rank (f.r.r.) and that is has more columns than rows, that is, we need more DOFs n than number of contacts n_c. This mapping would be *onto* (surjective) meaning that the whole space of motions in the constrained directions is reachable. However, the mapping is not be *one-to-one* (injective) since several joint velocities would map to the same contact twists.

As a dual point of view, it means that $\boldsymbol{J}_H^T$ is full column rank (f.c.r.) with more rows than columns. These means its range is not fully spanned (the mapping is not onto), but is one-to one.

Common methods for the analysis of these subspaces do not take into account that the snake robot is a mobile robot (i.e., it has non-actuated DOFs). Instead of analyzing $\boldsymbol{J}_H$ and $\boldsymbol{J}_H^T$ we will analyze ${}^a\boldsymbol{J}_H$, ${}^a\boldsymbol{J}_H^T$ directly. In this way, we will only analyzed the contributions of the joints of the snake robot,

The dimensions of the subspaces of ${}^a\boldsymbol{J}_H \in \Re^{n_c \times n_a}$ and ${}^a\boldsymbol{J}_H^T \in \Re^{n_a \times n_c}$ can be summarized as:

$$ {}^a\boldsymbol{J}_H, {}^a\boldsymbol{J}_H^T \begin{cases} \dim(\mathcal{R}({}^a\boldsymbol{J}_H)) = \text{rank}({}^a\boldsymbol{J}_H) \\ \dim(\mathcal{N}({}^a\boldsymbol{J}_H)) = n_a - \text{rank}({}^a\boldsymbol{J}_H) \\ \dim(\mathcal{R}({}^a\boldsymbol{J}_H^T)) = \text{rank}({}^a\boldsymbol{J}_H) \\ \dim(\mathcal{N}({}^a\boldsymbol{J}_H^T)) = n_c - \text{rank}({}^a\boldsymbol{J}_H) \end{cases} \tag{2.30} $$

The desired properties can be states as follows:

Objective 3

Redundant: Redundancy is desired, since it would allow the snake robot to have extra DOFs while maintaining contact with the object. Then, it is desired that $\mathcal{N}({}^a\boldsymbol{J}_H) \neq \mathbf{0}$. For that, it is necessary that $n_a > \text{rank}({}^a\boldsymbol{J}_H)$

Objective 4

Non-defective: In order to avoid hyper-static grasps and in general contact forces $\boldsymbol{\lambda}$ that could not be generated by the snake robot, it is desired that $\mathcal{N}({}^a\boldsymbol{J}_H^T) = \mathbf{0}$. In other words, it is required that $\text{rank}({}^a\boldsymbol{J}_H) = n_c$

Necessary and sufficient conditions are that ${}^a\boldsymbol{J}_H$ is full row rank (f.r.r.) and that it has more columns than rows, that is, we need more actuated DOFs n_a than number of contacts n_c. The

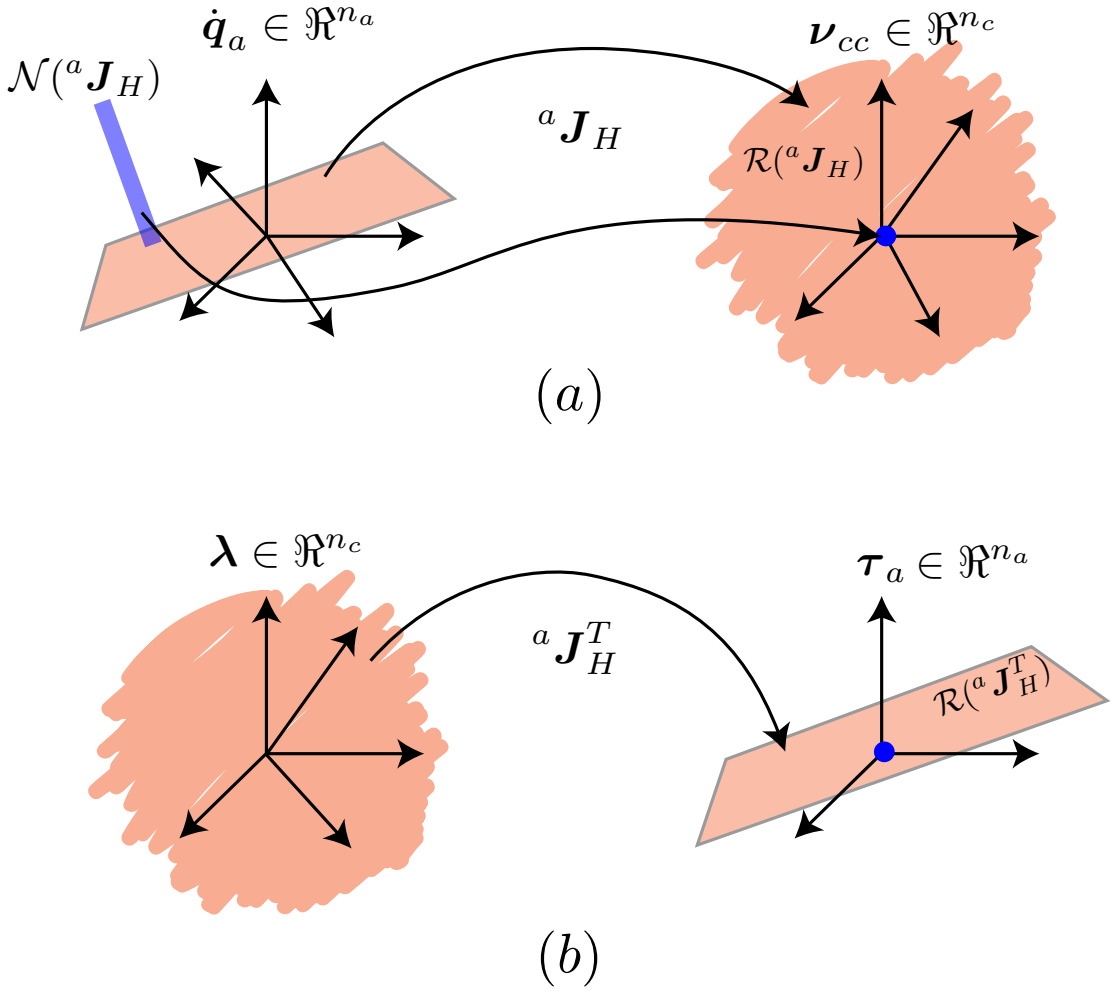


FIGURE 2.7: **Subspaces of ${}^a J_H$ and ${}^a J_H^T$**
(a) The mapping ${}^a J_H$ and its subspaces. Notice that $\mathcal{N}({}^a J_H)$ is nontrivial and the subspace of contact twists ν_{cc} is fully accessible. (b) The mapping ${}^a J_H^T$ and its subspaces. Notice that $\mathcal{N}({}^a J_H^T)$ is trivial. Any contact force will have an effect in the snake robot.

mapping would be surjective, but not injective. As a dual point of view, ${}^a J_H^T$ is full column rank (f.c.r.) with more rows than columns. The mapping is one-to-one, but not onto. These concepts are graphically shown in Fig. 2.7.

The conditions on ${}^a J_H$ and ${}^a J_H^T$ are more strict than if we had analyzed J_H and J_H^T. The conditions on ${}^a J_H$ to be f.r.r. (and at the same time ${}^a J_H^T$ is f.c.r.) imply that the constraints (2.22) must be independent. That is easy to accomplish for frictionless contact points as long as the contacts are not coincident. Furthermore, the condition $n_a > \text{rank}({}^a J_H)$ can be restated as $n_a > n_c$. This can be interpreted in several ways, but the easiest one is that, given a number of contacts n_c, at least $n_c + 1$ actuated joints are necessary.

Notice that, thanks to the fact that $\mathcal{N}({}^a J_H^T) \neq \mathbf{0}$ (and as a consequence $\mathcal{N}(J_H^T) \neq \mathbf{0}$), the grasp cannot be hyper-static even if $\mathcal{N}(G) = \mathbf{0}$, since $\mathcal{N}(J_H^T) \bigcap \mathcal{N}(G) = \mathbf{0}$

Remarks: Notice that thanks to the fulfillment of Objective 2 and Objective 4, it can be considered that a snake robot could grasp the object with form-closure. This can be further explained by

TABLE 2.1: Parameters of Example I

Parameter	Value	Description
n	6	Number of DOFs of the system
n_a	3	Number of actuated joints of the snake robot
n_a+1	4	Number of links of the snake robot
n_c	3	Number of frictionless contacts
$\boldsymbol{G}^T$	$\in \Re^{3\times 3}$	Transpose of grasp matrix, with its third column full of zeros
$\boldsymbol{G}$	$\in \Re^{3\times 3}$	Grasp matrix with its third row full of zeros
$\boldsymbol{J}_H$	$\in \Re^{3\times 6}$	Hand Jacobian (f.r.r.)
$\boldsymbol{J}_H^T$	$\in \Re^{6\times 3}$	Transpose of the hand Jacobian (f.c.r.)
$\dim(\mathcal{R}(\boldsymbol{G}^T))$	2	Dimensions of the column space of $\boldsymbol{G}^T$
$\dim(\mathcal{N}(\boldsymbol{G}^T))$	1	Dimensions of the nullspace of $\boldsymbol{G}^T$. The grasp is partially indeterminate
$\dim(\mathcal{R}(\boldsymbol{G}))$	2	Dimensions of the column space of $\boldsymbol{G}$
$\dim(\mathcal{N}(\boldsymbol{G}))$	1	Dimensions of the nullspace of $\boldsymbol{G}$. The grasp is graspable. There exists internal forces.
$\dim(\mathcal{R}({}^a\boldsymbol{J}_H))$	2	Dimensions of the column space of ${}^a\boldsymbol{J}_H$
$\dim(\mathcal{N}({}^a\boldsymbol{J}_H))$	1	Dimensions of the nullspace of ${}^a\boldsymbol{J}_H$. It is redundant. Motions of joint ϕ_6 do not affect the object.
$\dim(\mathcal{R}({}^a\boldsymbol{J}_H^T))$	2	dimensions of the column space of ${}^a\boldsymbol{J}_H^T$
$\dim(\mathcal{N}({}^a\boldsymbol{J}_H^T))$	1	dimensions of the nullspace of ${}^a\boldsymbol{J}_H^T$. It is defective.

looking at the constraint matrix $\boldsymbol{A}$ from 2.23. Since both $\boldsymbol{J}_H$ and $\boldsymbol{G}^T$ are both f.r.r., so it is $\boldsymbol{A}$. It is important to notice that this does not mean that the object could be fully constrained with respect to (w.r.t.) the inertial frame. It is, however, grasped w.r.t. the snake robot. To guarantee that the object is fully immobilized w.r.t. the base, a dynamic (or quasi-static) analysis would be necessary, however, this would be an analysis of force-closure. Grasps with force-closure are beyond the scope of this paper. However, they are certainly interesting and are considered our immediate future research. The main difficulty is that the contribution of the actuated joints ($\boldsymbol{\tau}_a$) does not affect directly the non-actuated DOFs. In other words, the analysis of form-closure cannot distinguish the lack of a fixed-base. This makes it obvious that a snake robot would need the help of external forces (i.e., friction acting on the links) in order to fully constrain and manipulate the object.

2.3.4 Examples

So far we have given necessary conditions for a snake robot to grasp with form-closure an object. Next, we present two examples in order to understand these concepts better.

Example I: Partially Indeterminate, Graspable, Redundant, and Defective Grasp with Partial Form-closure

In this example, we consider a snake robot with three links grasping an object. Fig. 2.8 shows the system considered. The contacts occur at links: $link_3$, $link_4$, and $link_5$. The general properties of the system are enumerated in Table 2.1. Furthermore, the contents of the matrix ${}^a\boldsymbol{J}_H^T \in \Re^{3\times 2}$ are shown. Remember these mapping takes into account only the actuated DOFs of the snake robot:

$$ {}^a\boldsymbol{J}_H = \begin{bmatrix} 0 & 0 & 0 \\ \hat{\boldsymbol{z}}_{c2}^T \boldsymbol{d}_{4,4} & 0 & 0 \\ \hat{\boldsymbol{z}}_{c3}^T \boldsymbol{d}_{5,4} & \hat{\boldsymbol{z}}_{c3}^T \boldsymbol{d}_{5,5} & 0 \end{bmatrix} \in \Re^{3\times 2} \tag{2.31} $$

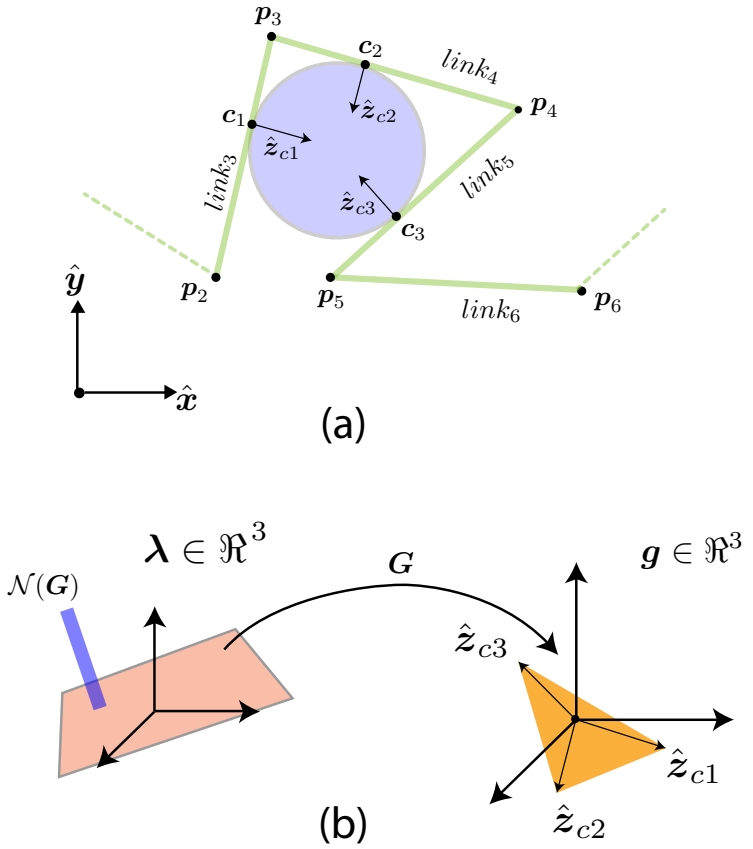


FIGURE 2.8: **Examples grasp: partially indeterminate, graspable, redundant, defective**
(a) System considered in Example I. The grasp is defective because the object is contacting $link_3$ which doesn't have an actuator. The link could be thought of as the *palm* of a robotic hand. (b) Notice that $\mathcal{N}(G)$ is nontrivial, therefore there are internal forces, and are controllable by the snake robot. Also notice that the space g is not fully accessible. This is because the snake robot cannot exert a torque unto the object. However, there is partial form-closure since the axes $\hat{z}_{ck}$ positively span a plane.

where,

$$d_{4,4} = \hat{z}_3 \times (c_2 - p_3)$$

$$d_{5,4} = \hat{z}_3 \times (c_3 - p_3)$$

$$d_{5,5} = \hat{z}_4 \times (c_3 - p_4).$$

It is interesting to see that the structure of ${}^a J_a$ could change if the object were contacting other links. If it were contacting $link_4$, $link_5$, and $link_6$ the grasp would not be defective.

Example II: Partially Indeterminate, Graspable, Redundant, Non-defective Grasp with Partial Form-closure

In this example, we consider a snake robot with three links and three contacts grasping an object. Fig. 2.9 shows the system considered. The contacts occur at links: $link_5$, $link_6$, and $link_7$. The

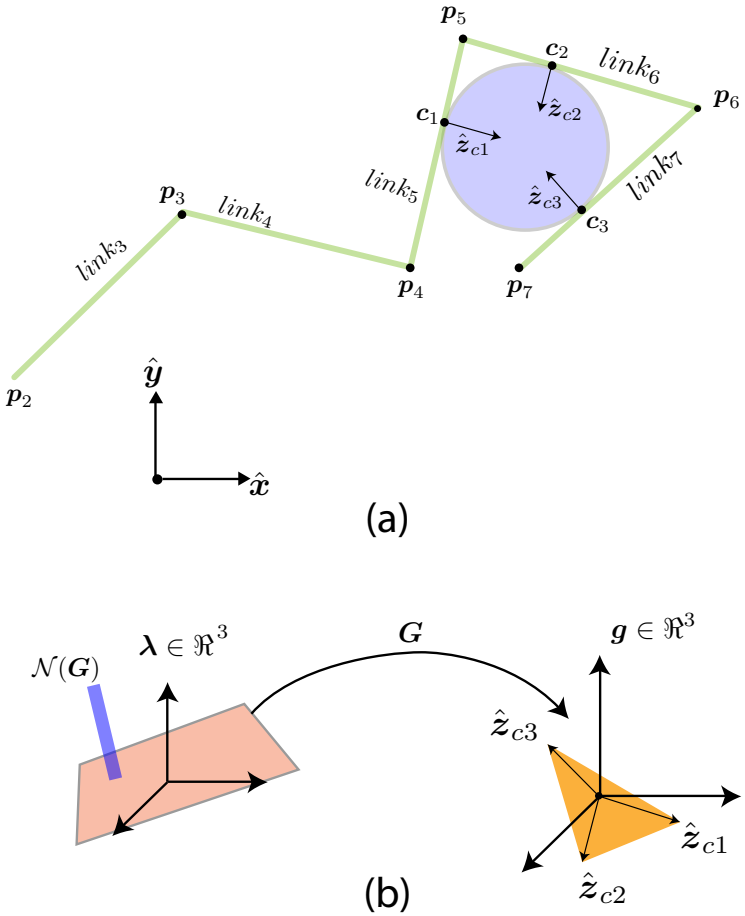


FIGURE 2.9: **Examples grasp: partially indeterminate, graspable, redundant, non-defective**
(a) System considered in Example II. (b) Notice that $\mathcal{N}(\boldsymbol{G})$ is non-trivial, therefore there are internal forces and are controllable by the snake robot. Also notice that the space $\boldsymbol{g}$ is not fully accessible as in Example I.

general properties of the system are enumerated in Table 2.2 Furthermore, the general shape of the matrix ${}^{a}\boldsymbol{J}_H^T \in \Re^{3\times 4}$ is shown.

$$ {}^{a}\boldsymbol{J}_H = \begin{bmatrix} * & * & 0 & 0 \\ * & * & * & 0 \\ * & * & * & * \end{bmatrix} \in \Re^{3\times 4} \tag{2.32} $$

The properties of $\boldsymbol{G}^T$ and $\boldsymbol{G}$ do not change because they do not depend on the structure of the mechanism used to grasp the object.

2.3.5 Conclusions Regarding Types of Grasps with Snake Robots

In this section, we have presented a analysis of grasping with snake robots. We have shown that several properties can be understood through analyzing the kinematic structure of the system

TABLE 2.2: Parameters of Example II

Parameter	Value	Description
n	7	Number of DOFs of the system
n_a	4	Number of actuated joints of the snake robot
$n_a + 1$	5	Number of links of the snake robot
n_c	3	Number of frictionless contacts
$\boldsymbol{G}^T$	$\in \Re^{3\times 3}$	Transpose of grasp matrix, with its third column full of zeros
$\boldsymbol{G}$	$\in \Re^{3\times 3}$	Grasp matrix with its third row full of zeros
$\boldsymbol{J}_H$	$\in \Re^{3\times 7}$	Hand Jacobian (f.r.r.)
$\boldsymbol{J}_H^T$	$\in \Re^{7\times 3}$	Transpose of the hand Jacobian (f.c.r.)
$\dim(\mathcal{R}(\boldsymbol{G}^T))$	2	Dimensions of the column space of $\boldsymbol{G}^T$
$\dim(\mathcal{N}(\boldsymbol{G}^T))$	1	Dimensions of the nullspace of $\boldsymbol{G}^T$. The grasp is partially indeterminate
$\dim(\mathcal{R}(\boldsymbol{G}))$	2	Dimensions of the column space of $\boldsymbol{G}$
$\dim(\mathcal{N}(\boldsymbol{G}))$	1	Dimensions of the nullspace of $\boldsymbol{G}$. The grasp is graspable. There exists internal forces.
$\dim(\mathcal{R}({}^a\boldsymbol{J}_H))$	3	Dimensions of the column space of ${}^a\boldsymbol{J}_H$
$\dim(\mathcal{N}({}^a\boldsymbol{J}_H))$	1	Dimensions of the nullspace of ${}^a\boldsymbol{J}_H$. It is redundant.
$\dim(\mathcal{R}({}^a\boldsymbol{J}_H^T))$	3	dimensions of the column space of ${}^a\boldsymbol{J}_H^T$
$\dim(\mathcal{N}({}^a\boldsymbol{J}_H^T))$	0	dimensions of the nullspace of ${}^a\boldsymbol{J}_H^T$. It is not defective.

with tools of linear algebra. The conditions presented are necessary, but not sufficient, for a snake robot to grasp an object with form-closure.

The main contributions of this section can be summarized as follows:

1. *Requirements for partially indeterminate and graspable grasps*: At least three frictionless contact points are necessary ($n_c \geqslant 3$). The position of the object (but not orientation) is constrained.

2. *Requirements for redundant and non-defective grasps*: It is necessary to have more joints than contacts ($n_a > n_c$), i.e., a snake robot with five links. Also, it is necessary that the first contacting link is not $link_3$.

We have not limited our study to the analysis of $\boldsymbol{G}^T$ and $\boldsymbol{G}$ because it is important to consider the structure of the snake robot in the analysis. Furthermore, we extended the analysis to only consider the actuated joints of the snake robot. This is not usually done, since regularly the analysis is done w.r.t. to the *palm* of a robotic hand, or any other fixed point on the robot. These properties can be better understood by analyzing and comparing the mappings $\boldsymbol{J}_H$, $\boldsymbol{J}_H^T$, ${}^a\boldsymbol{J}_H$, and ${}^a\boldsymbol{J}_H^T$.

Form-closure, as it has been presented in this paper, would mean that the object is grasped w.r.t. the snake robot. However, this is not sufficient for manipulating the object, since the snake robot does not have a fixed-base and it is not realistic to expect that it can exert arbitrary wrenches unto the object. In order to analyze manipulation with snake robots a quasi-static or dynamic analysis would be required, and to test for force-closure.

In the following section, a set of necessary geometric conditions are presented, that have to be met for a snake robot to grasp an object with (partial) form-closure.

2.4 Form-Closure - Feasible Solutions

A grasp is said to have form-closure when the motions of the object can be restrained by the robot. In other words, the robot could stop the object from moving given an external disturbance. Form-closure does not take friction into account, which makes a more secure grasp [37, 63]. More specifically, this thesis focuses on the analysis (first-order) of form-closure, that is, we do not take into consideration the curvature of the surfaces in contact.

Form-closure has some advantages. For example, it can be calculated faster since it relies solely on geometrical analysis. Force closure-on the other hand, requires to take into account forces, and consequently it is necessary to consider the forces that a specific robot can exert [37, 39, 64]. A disadvantage of form-closure is that is usually requires more contact points to guarantee a successful grasp, compared to an analysis using force-closure.

2.4.1 Partial form-closure

Grasps with form-closure have some limitations. As it has been mentioned in [35] and proven in [65], exceptional surfaces like solids of revolution and circles in the plane can never be fully restrained with frictionless contacts. In other words, only the translation but not the rotation of the object can be restrained, achieving only *partial* form-closure. The minimum number of contacts necessary to achieve partial form-closure in a plane is three frictionless contacts.

In the remainder of this thesis, the term *partial* will be dropped, with the understanding that only translation can be restrained. Another reason to analyze the case of form-closure is that it can be easily extended to the concept of *caging* [66].

A complete proof of existence of form-closure is beyond the scope of this thesis and has been thoroughly researched by other authors. The references [35] and [37] are recommended for a deeper study of the topic. In the following, a test is described that is conventionally used to test for form-closure and to obtain a measure of the quality of the grasp [37]. This test can be formulated as a linear program (LP) optimization problem. Assume a force $\lambda_k \geqslant 0$ can be exerted at the contact on the $\hat{\boldsymbol{z}}_{ck}$ direction; such forces are collected on the vector $\boldsymbol{\lambda} \in \Re\, n_c \times 1$. Denote with $d \in \Re$ the smallest component of $\boldsymbol{\lambda}$. The purpose of the LP is to maximize the quantity d. The optimal value obtained d^* is bounded as $0 \leqslant d^* \leqslant 1$. If $d^* > 0$ then there is form-closure, and the lower the value the closer the grasp is to lose form-closure. If $d^* = 0$ then there is not form-closure. Obviously, grasps with high values of d^* are desired. Therefore, the value d^* can be used as a metric to analyze a given grasp. The interpretation of this test is to check if the object can be squeezed by the robot, without producing movement. This is equivalent to check that the null-space of the forces is not trivial.

This LP problem can be solved on real-time. However it may be computationally expensive, or ill-conditioned since we may not know the conditions *a priori*. In this section, we show that we can exploit our knowledge of the kinematic structure of the snake robot and propose a simplified form-closure test that rely purely on geometric insight. This is inspired in other proposed tests,

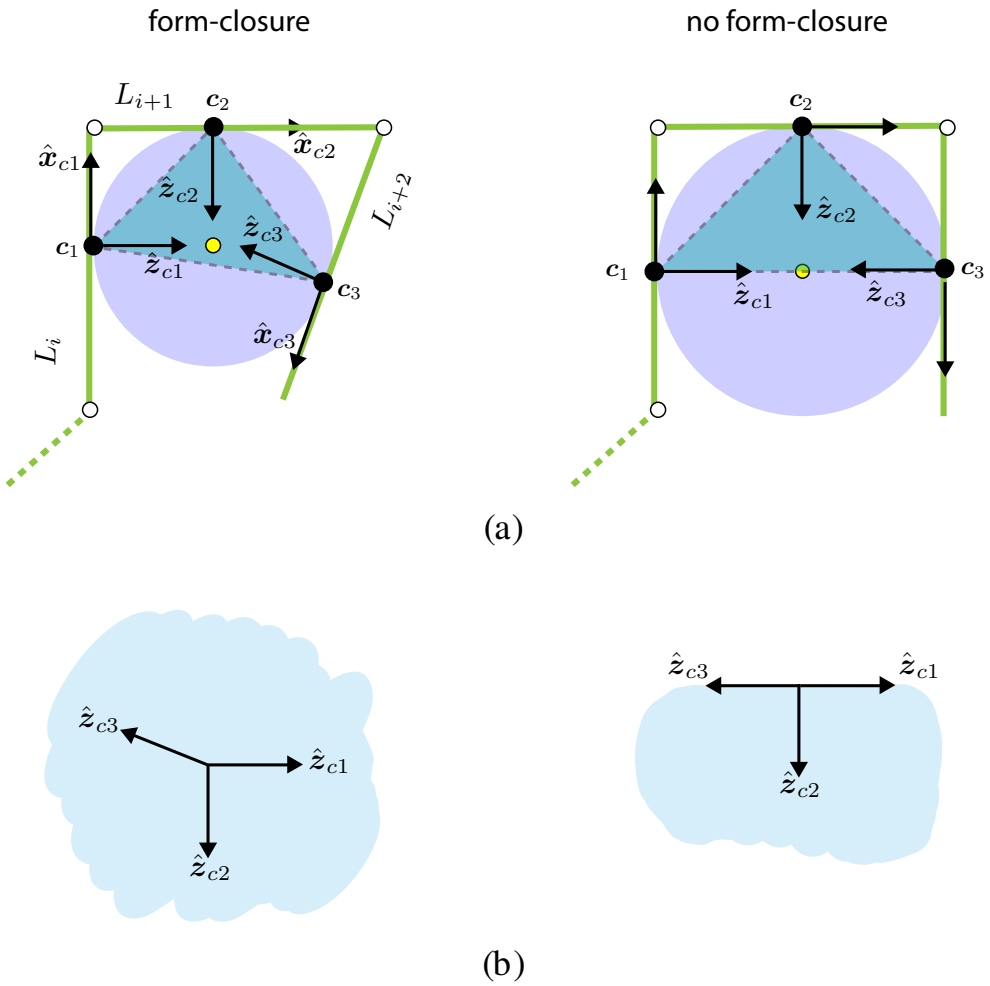


FIGURE 2.10: **Form-closure test with enveloping grasp**
Three contacts between three adjacent links of the snake robot and an object. The COM of the object is shown as a yellow circle. The polygon described by the contact points is also shown. (a) The COM of the object is inside or outside the polygon and the axes $\hat{z}_{ck}$ (b) The axes $\hat{z}_{ck}$ span the whole space when there is form-closure

like [59], which basically states that if the contact forces span the whole space, there is form-closure.

Figure 2.10 shows two intuitive ways to test for form-closure. If the snake robot is contacting an object, the contact points form a polygon. In the specific case of three contact points, the polygon is a triangle. Given this triangle, the position of the COM of the object can be calculated and if it is strictly inside the triangle (by strictly inside we mean it cannot be on the boundary), then there is form-closure. This test can be seen in Fig. 2.10(a), where the same snake robot grasps two objects of different size. There is an equivalent interpretation when talking about forces. At the contact points, the vectors normal to the snake robot $\hat{z}_{ck}$ span the subspace of forces that can be imparted on the object. If these spanning vectors span the plane, then there is *form-closure*. This test can be seen in Fig. 2.10(b).

2.4.2 Grasping with three adjacent links: grasp condition $\mathcal{G}^3$

This section focuses on the analysis of form-closure using an enveloping grasp under the specific situation that the snake robot is contacting the object with three adjacent links. The objective is to analyze if form-closure is possible or not. Additionally, the set of circumstances under which form-closure is possible is analyzed, providing a quick test for form-closure. This test is binary, in other words, it qualifies if a grasp with form-closure is possible or not.

The snake robot and object to be grasped first have to be modeled. This model is purely geometric, allowing to gather well-grounded conclusions. The main parameters to analyze the problem are:

- The length of the links ℓ. It is assumed all links have the same length.
- The ratio between the radius of the object and the length of the links, denoted as r_μ.
- The position of the contact point on the first contacting link $\Delta\ell$.

The ratio between the radius of the object and the length of the links r_μ is defined as

$$r_\mu := r_{obj}/\ell. \tag{2.33}$$

By using the ratio r_μ we can focus on the relationship of size between the robot and object, allowing to study any case at the same time. Fig. 2.11(a) shows a visual representation of all involved quantities.

Under the assumption that at least three frictionless point contacts, i.e. $n_c = 3$, are necessary to form-close the object and that each link can only contact the object at one point, it is concluded that at least three links are necessary to form-close the object. This has also been discussed in Section 2.3.5. Furthermore, let's assume that the first link contacting the object is called L_i, and the next two adjacent links are L_{i+1} and L_{i+2}. It can be seen in Fig. 2.11(a) that there are three general cases, depending on the size of the object w.r.t. the snake robot's link.

Fig. 2.11(b) shows the geometry involved in the grasp. The distance from the joint connecting the first and second contacting links to the first contact point is denoted by $\Delta x \in \Re$, $\Delta x \geqslant 0$. If $\Delta x = 0$ or $\Delta x = \ell$ then only two contact points exist, even if there are three contacting links. It can be concluded that a grasp with form-closure is not possible. This would mean the object is contacting the snake robot at the joint. These cases can be see in 2.11(b.A) and (b.B) The cases where $0 < \Delta x < \ell$ is then what we are interested in, as it can be seen in 2.11(b.C).

There are two types of triangles that can be formed that need to be investigated. The triangles $\triangle \boldsymbol{p}_{obj}\boldsymbol{c}_1\boldsymbol{p}_i$ and $\triangle \boldsymbol{p}_{obj}\boldsymbol{p}_i\boldsymbol{c}_2$ are created when links L_i and L_{i+1} contact the object. These two triangles are always identical, so their inner angles α_1 and α_2 must also be identical. The angle α can be defined as

$$\alpha := \alpha_1 = \alpha_2 = \tan^{-1}\left(\frac{\Delta x}{r_{obj}}\right). \tag{2.34}$$

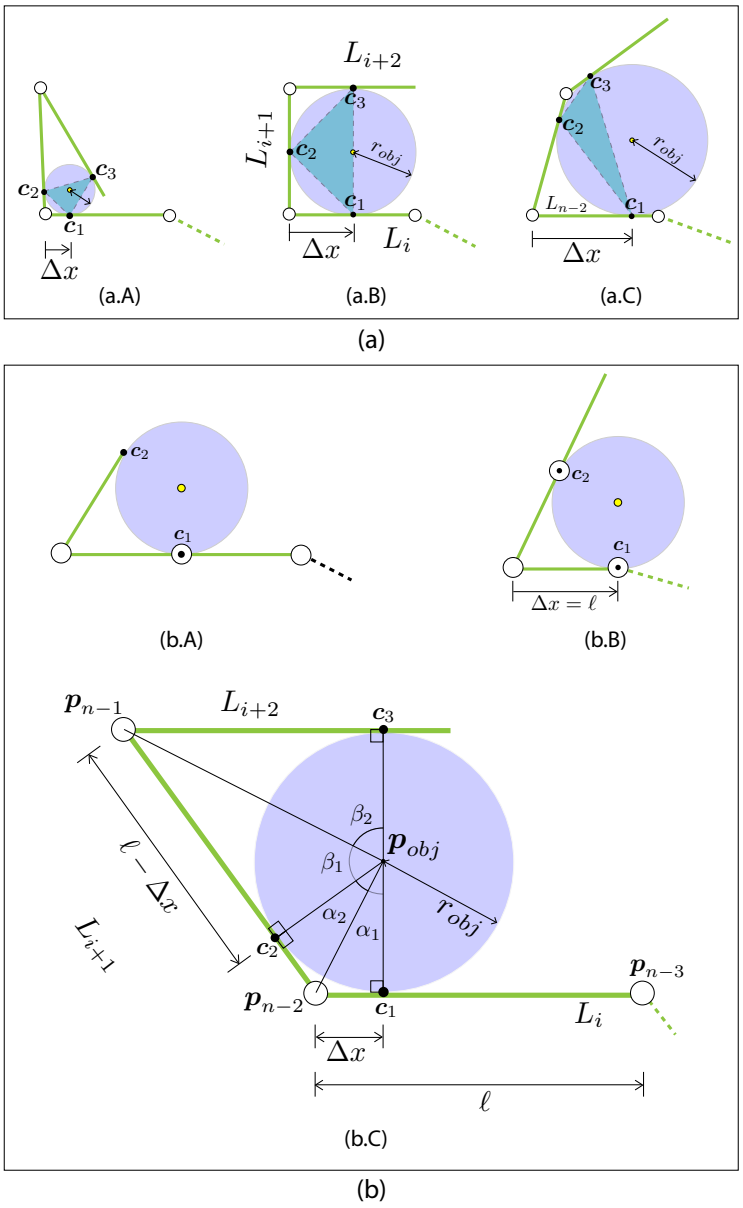


FIGURE 2.11: **Form-closure region**
(a) Three representative cases are marked: (a.A) Inside the region ($\Delta x = 0.3, r_\mu = 0.2$). There is form-closure, (a.B) On the boundary ($\Delta x = 0.5, r_\mu = 0.5$). There is not form-closure. (a.C) Outside the region ($\Delta x = 0.8, r_\mu = 0.6$). There is not form-closure. (b) The geometric analysis of the grasp

The next two triangles $\triangle \boldsymbol{p}_{obj}\boldsymbol{c}_2\boldsymbol{p}_{i+1}$ and $\triangle \boldsymbol{p}_{obj}\boldsymbol{p}_{i+1}\boldsymbol{c}_3$ are created when the links L_{i+1} and L_{i+2} contact the object. Similarly as before, their inner angles are identical and can be defined as

$$\beta := \beta_1 = \beta_2 = \tan^{-1}\left(\frac{\ell - \Delta x}{r_{obj}}\right) \tag{2.35}$$

From the requirements of form-closure presented in previous sections, it can be intuitively stated that a sufficient condition for the existence of form-closure is that $2\alpha + 2\beta > 180°$. Otherwise, the COM of the object denoted by $\boldsymbol{p}_{obj}$ would be outside the polygon formed by $\triangle \boldsymbol{c}_1\boldsymbol{c}_2\boldsymbol{c}_3$. This

condition can be described mathematically as

$$2\alpha + 2\beta > 180^\circ, \tag{2.36}$$

and represented as a function of the parameters Δx, ℓ and r_{obj} as

$$\tan^{-1}\left(\frac{\Delta x}{r_{obj}}\right) + \tan^{-1}\left(\frac{\ell - \Delta x}{r_{obj}}\right) > 90^\circ, \tag{2.37}$$

$$\tan^{-1}\left(\frac{r_{obj}\ell}{{r_{obj}}^2 - \Delta x\ell + \Delta x^2}\right) > 90^\circ \tag{2.38}$$

After some manipulation of the above expression, the following inequality is obtained:

$$r_\mu < \sqrt{\frac{\Delta x\ell - \Delta x^2}{\ell^2}} \tag{2.39}$$

where the inequality is expressed using the scale invariant ratio (2.33). The definition of the grasping condition follows naturally as:

$$\mathcal{G}^3 := \{(r_\mu, \Delta x): \quad r_\mu < \sqrt{\frac{\Delta x\ell - \Delta x^2}{\ell^2}}, 0 < \Delta x < \ell\} \tag{2.40}$$

The set $\mathcal{G}^3$ describes the region in the $\Delta x - r_\mu$ plane where form-closure with three frictionless contact points is possible, as shown in Fig. 2.12(a). Since the quantity r_μ is scale invariant and the displacement Δx is constrained to lie in the range $0 \leqslant \Delta x \leqslant \ell$, a snake robot with unit length links is considered ($\ell = 1$) without regards of the units. In the same figure, three representative examples are shown, which can be compared to the ones in Fig. 2.11(a). If the relative size between the object and snake robot fulfills the grasp condition $\mathcal{G}^3$ then the COM of the object will be inside the polygon described by the contact points, and form-closure will be possible with three adjacent links.

The grasp condition (2.40) answers two very important questions. If the first contact location on link L_i is known (denoted by Δx_0), the range of objects that can be grasped is easily obtained:

$$0 < r_\mu < \sqrt{\frac{\Delta x_0\ell - {\Delta x_0}^2}{\ell^2}} \tag{2.41}$$

or more interestingly, given an object with certain radius (denoted by $r_{\mu 0}$) it is possible to analyze the range of locations where the first contact point on link L_i must lie:

$$\Delta x_1 < \Delta x < \Delta x_2 \tag{2.42}$$

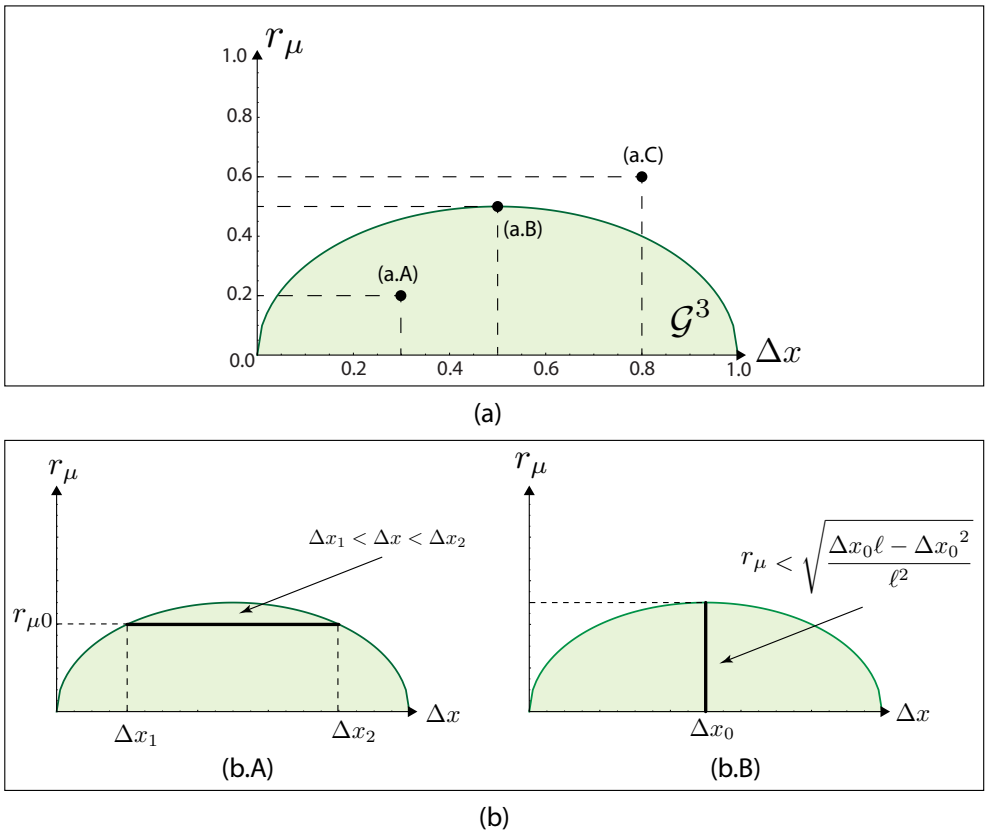


FIGURE 2.12: **Form-closure evaluation**
(a) The grasp region where form-closure is possible (cf. Fig. 2.11(a)) (b.A) Given a certain object with relative size $r_{\mu 0}$, the range where the first contact point can lie on the first contacting link can be found with (2.42). (b.B) Given a contact location Δx_0, objects of several sizes can be grasped, as described by (2.41)

where the lower and upper limits can be calculated as

$$\Delta x_1 = \frac{\ell}{2} - \frac{1}{2}\sqrt{\ell^2 - 4\ell^2 {r_{\mu 0}}^2}$$

$$\Delta x_2 = \frac{\ell}{2} + \frac{1}{2}\sqrt{\ell^2 - 4\ell^2 {r_{\mu 0}}^2}.$$

Both concepts are shown in Fig. 2.12(b).

2.5 (Unconstrained) Dynamic Modeling of the Snake Robot

Previous sections have concentrated in the geometric (kinematic) analysis of a snake robot contacting an object. In the following, the analysis is extended to consider the inertial parameters of the system. The equations of motion of the snake robot can be presented in the canonical form

$$\boldsymbol{M}_s \ddot{\boldsymbol{q}}_s + \boldsymbol{h}_s = \boldsymbol{B}\boldsymbol{\tau}_{act} + \boldsymbol{\tau}_{ext} \tag{2.43}$$

where $\boldsymbol{M}_s(\boldsymbol{q}_s) \in \Re^{n_s \times n_s}$ is the inertia matrix of the snake robot (a symmetric positive definite (PD) matrix), $\boldsymbol{h}_s(\boldsymbol{q}_s, \dot{\boldsymbol{q}}_s) \in \Re^{n_s \times 1}$ contains Coriolis and centripetal effects, and $\boldsymbol{\tau}_{ext}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \Re^{n_s \times 1}$ is a vector of torques produced by external forces (e.g., kinetic friction). The matrix $\boldsymbol{B} \in \Re^{n_s \times n_a}$ defined as

$$\boldsymbol{B} := \begin{bmatrix} \boldsymbol{0}_{3 \times n_a} \\ \boldsymbol{1}_{n_a \times n_a} \end{bmatrix}, \tag{2.44}$$

is a selection matrix that projects the vector of input forces $\boldsymbol{\tau}_{act} \in \Re^{n_a \times 1}$ into the space of generalized forces. The matrix $\boldsymbol{1}$ denotes the identity matrix of appropriate dimensions. As it can be seen from model (2.43), the snake robot is underactuated since the actuators do not have a direct input into the first three rows of (2.43). If all links have the same mass ($m_i = m_\ell \quad \text{for } i = 1, ..., n_\ell$), then the inertia matrix can be factorized as

$$\boldsymbol{M}_s = m_s \bar{\boldsymbol{M}}_s = \frac{m_T}{n_\ell} \bar{\boldsymbol{M}}_s, \tag{2.45}$$

where $\bar{\boldsymbol{M}}_s$ is an inertia matrix where all links have unitary mass.

The plane where the system lies depends on the angle θ_{slope} (c.f. Fig. 2.4 (c)). Part of the gravity effects $\boldsymbol{g}$ act parallel to the plane $\boldsymbol{g}_t = (1 - \cos(\theta_{slope}))\boldsymbol{g}$, and the normal component is $\boldsymbol{g}_n = \cos(\theta_{slope})\boldsymbol{g}$. The generalized forces resulting from the gravity effects $\boldsymbol{\tau}_g$ can be obtained as

$$\boldsymbol{\tau}_g = \sum_{i=1}^{n_\ell} m_i \boldsymbol{J}_i^T \boldsymbol{g}_t. \tag{2.46}$$

If gravity is the only external force acting on the system, then $\boldsymbol{\tau}_{ext} = \boldsymbol{\tau}_g$ and will vanish for the horizontal plane.

In this thesis, we assume that the snake robot is composed of two distinct sections with a different objective. The *locomotion section* of the snake robot have passive wheels and additional constraints due to (static) friction, that will be introduced in Appendix A.4.2, need to be added to the model. The *manipulation section* does not have passive wheels and will be used to manipulate an object. This idea is equivalent to the model presented in [25] and in [19]. Mathematically, it could also be linked to [44], where constraints between the snake robot and ground are removed in order to gain controllability.

2.6 Coupled Dynamic Model Between Snake Robot and Object

If the system were composed of two bodies B_1 and B_2, the equations of motion (now coupled with the constraints) could be compactly written as

$$m_1 \bar{\boldsymbol{I}} \boldsymbol{a} + \boldsymbol{p} = \boldsymbol{f}_{act} + \boldsymbol{f}_{ext} + \boldsymbol{f}_c \tag{2.47}$$

$$\bar{\boldsymbol{I}} := \begin{bmatrix} \bar{\boldsymbol{I}}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \kappa\bar{\boldsymbol{I}}_2 \end{bmatrix} \qquad \boldsymbol{a} := \begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \end{bmatrix} \qquad \boldsymbol{p} := \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \end{bmatrix}$$

$$\boldsymbol{f}_{act} := \begin{bmatrix} \boldsymbol{f}_{act1} \\ \boldsymbol{f}_{act2} \end{bmatrix} \qquad \boldsymbol{f}_{ext} := \begin{bmatrix} \boldsymbol{f}_{ext1} \\ \boldsymbol{f}_{ext2} \end{bmatrix} \qquad \boldsymbol{f}_c = \boldsymbol{A}^T\boldsymbol{\lambda}$$

while considering the second order constraints

$$\boldsymbol{A}\boldsymbol{a} + \dot{\boldsymbol{A}}\boldsymbol{v} \geqslant \boldsymbol{0}. \tag{2.48}$$

The terms $\boldsymbol{f}_{acti}$ and $\boldsymbol{f}_{exti}$ for $i = 1, 2$ express the input (actuated) forces and external acting on each system, respectively.

Although this is a simplification containing only two bodies, the extension to a snake robot contacting an object follows naturally, and are shown in more detail in Section 2.7.3. The eqns. of motion (2.47) exploit the relationship between the masses in the system. The mass of B_2 can be expressed as proportional to the total mass of B_1 as

$$m_2 = m_1\kappa, \quad \kappa > 0. \tag{2.49}$$

More specifically, when dealing with a snake robot in contact with an object, the previous equation can be written as

$$m_{obj} = m_s\kappa, \quad \kappa > 0. \tag{2.50}$$

This ratio allows us to study several systems at once. The equations show clearly that, it is not really the mass of the object that is important, but its relationship with respect to the mass of the snake robot.

The equations of motion of the snake robot (2.43) would replace the equations of B_1 and the object would be represented by B_2. All the constraints can be put together

$$\boldsymbol{A}\begin{bmatrix} \dot{\boldsymbol{q}}_s \\ \boldsymbol{v}_{obj} \end{bmatrix} \geqslant \boldsymbol{0} \tag{2.51}$$

where the constraint matrix $\boldsymbol{A} \in \Re^{n_c \times (n_{op}+n_s)}$ takes the following form

$$\boldsymbol{A} = \begin{bmatrix} -\boldsymbol{J}_s & \boldsymbol{G}^T \end{bmatrix}. \tag{2.52}$$

The matrix $\boldsymbol{J}_s \in \Re^{n_c \times n_s}$ is called the robot Jacobian (also called *hand Jacobian* [35, 37]) which projects the vector of generalized velocities of the snake robot onto the subspace spanned by the constraints. The matrix $\boldsymbol{G} \in \Re^{n_{op} \times n_c}$ is usually referred to as *Grasp Matrix* and its transpose is a mapping from the motion space of the object to the constrained subspace; it can be constructed in a similar manner to the robot Jacobian. The constraint forces projected back onto the snake robot and object are

$$\begin{bmatrix} \boldsymbol{\tau}_c \\ \boldsymbol{f}_c \end{bmatrix} = \begin{bmatrix} -\boldsymbol{J}_s^T \\ \boldsymbol{G} \end{bmatrix} \boldsymbol{\lambda} = \boldsymbol{A}^T\boldsymbol{\lambda}, \tag{2.53}$$

where $\boldsymbol{\tau}_c \in \Re^{n_s}$ is simply the projection of the constraint forces onto the space of generalized forces of the snake robot and $\boldsymbol{f}_c \in \Re^{n_{op}}$ is the wrench acting on the object due to the constraints.

Both the Hand Jacobian and the Grasp Matrix have been thoroughly researched in the context of grasping. However, as it can be understood by comparing (2.53) to (A.6) and (A.7), both matrices are nothing more than a change of coordinates.

The more complex equations of motion can be written as

$$m_\ell \bar{\boldsymbol{I}} \boldsymbol{a} + \boldsymbol{p} = \boldsymbol{f}_{act} + \boldsymbol{f}_{ext} + \boldsymbol{A}^T \boldsymbol{\lambda}, \tag{2.54}$$

$$\bar{\boldsymbol{I}} = \begin{bmatrix} \bar{\boldsymbol{M}}_s & \boldsymbol{0} \\ \boldsymbol{0} & \kappa \bar{\boldsymbol{I}}_{obj} \end{bmatrix} \qquad \boldsymbol{a} = \begin{bmatrix} \ddot{\boldsymbol{q}}_s \\ \boldsymbol{a}_{obj} \end{bmatrix} \qquad \boldsymbol{p} = \begin{bmatrix} \boldsymbol{h}_s \\ \boldsymbol{p}_{obj} \end{bmatrix}$$

$$\boldsymbol{f}_{act} = \begin{bmatrix} \boldsymbol{B}\boldsymbol{\tau}_{act} \\ \boldsymbol{0} \end{bmatrix} \qquad \boldsymbol{f}_{ext} = \begin{bmatrix} \boldsymbol{\tau}_g \\ \kappa m_s \boldsymbol{g}_t \end{bmatrix}$$

where (2.54) exploits the relationship $m_{obj} = m_s \kappa$ discussed in (2.50) which exists when all links of the snake robot have the same mass. If a ratio with the total mass of the snake robot m_T is necessary, then the relationship $m_T = n_\ell m_s$ can be used, and the ratio takes the form

$$m_{obj} = \frac{m_T}{n_\ell} \kappa \tag{2.55}$$

However, since it is the term m_s which appears naturally in the factorization of the inertia tensor of the snake robot, the ratio (2.50) will be used.

This set of equations is standard in literature regarding multi-body systems (e.g., [33, 34]). However, the manipulation, analysis, and interpretation of such equations depend greatly on the system and task to be analyzed. The objective of this paper is to show the properties of such equations pertaining to a snake robot manipulating an object. This will place snake robots in a consistent framework with other robotic systems, making more clear the similarities and differences which may have been omitted in previous research (as discussed in Chapter 1).

Equations (2.54) have been presented in [24, 25, 26], but without a further analytical analysis. The simplified equations of motion (2.47) will be used for the analysis and interpretation of the equations. The inertial effects of the snake robot will be characterized by the position of its COM, which will change depending on the configuration. This will allows us to obtain conclusions that apply to any snake robot, regardless of the number of joints.

2.7 Projection onto the Constrained and Unconstrained Spaces

2.7.1 Constrained Subspace

By solving for $\boldsymbol{a}$ in (2.54) and substituting in (2.48), the constraint forces can be determined as

$$\boldsymbol{G}_c\boldsymbol{\lambda} = -\boldsymbol{A}\boldsymbol{I}^{-1}(\boldsymbol{f}_{act} + \boldsymbol{f}_{ext} - \boldsymbol{p}) + \dot{\boldsymbol{A}}\boldsymbol{v}, \tag{2.56}$$

where the matrix $\boldsymbol{G}_c \in \Re^{c\times c}$ defined as

$$\boldsymbol{G}_c := \boldsymbol{A}\boldsymbol{I}^{-1}\boldsymbol{A}^T = \frac{1}{m_s}\boldsymbol{A}\bar{\boldsymbol{I}}^{-1}\boldsymbol{A}^T = \frac{1}{m_s}\bar{\boldsymbol{G}}_c \tag{2.57}$$

is the metric tensor for the constraint forces and the matrix $\bar{\boldsymbol{G}}_c := \boldsymbol{A}\bar{\boldsymbol{I}}^{-1}\boldsymbol{A}^T$ considers the unitary inertia tensor $\bar{\boldsymbol{I}}$. It is a symmetric positive semi-definite (PSD) matrix and if it has full-rank (all the constraints are linearly independent) then it will be positive-definite (PD) and its inverse is defined as $\boldsymbol{G}_c^{-1} = m_s\boldsymbol{\Phi}$, where the matrix $\boldsymbol{\Phi} := \bar{\boldsymbol{G}}_c^{-1}$ can be interpreted as the inverse inertia of the system (projected onto the subspaces spanned by the constraints).

Multiplying to the left both sides of (2.56) by $\boldsymbol{G}_c^{-1}$ and taking into account (2.57), the constraint forces can be obtained as

$$\boldsymbol{\lambda} = \boldsymbol{\lambda}_{act} + \boldsymbol{\lambda}_{bias}, \tag{2.58}$$

$$\boldsymbol{\lambda}_{act} := -\boldsymbol{\Phi}\boldsymbol{A}\bar{\boldsymbol{I}}^{-1}\boldsymbol{f}_{act}, \tag{2.59}$$

$$\boldsymbol{\lambda}_{bias} := -\boldsymbol{\Phi}(\boldsymbol{A}\bar{\boldsymbol{I}}^{-1}(\boldsymbol{f}_{ext} - \boldsymbol{p}) - m_\ell\dot{\boldsymbol{A}}\boldsymbol{v}). \tag{2.60}$$

The affine system (2.58) represents a mapping from the inputs $\boldsymbol{f}_{act}$ to the constraint forces, with a bias term produced by velocity-induced terms and gravity.

The mapping (2.59) shows that the contribution from the input forces to the constraint forces does not depend explicitly on the masses of both systems, but on their ratio $\kappa = m_2/m_1$ encoded in $\bar{\boldsymbol{I}}$ (c.f. (2.47)). This mapping can be interpreted in several ways. In the context of robotics, the most common and useful interpretation is as a mapping from a quadratic region in the input space, onto the output space of constraint forces. This interpretations is commonly referred to as *force ellipsoid* [67, 68, 37, 28], where the origin will be shifted due to the bias terms. However, the previous analysis usually relies on a mapping from input (joint torques or velocities) to the force or velocity of an end-effector, without additional constraints. These results could not be applied directly to a snake robot since there are additional constraint forces and constraints (e.g., unilateral constraints for rigid-body contact and friction limits).

Since we are mainly interested in the contributions from the inputs to the constraint forces, the following linear mapping can be proposed

$$\Delta\boldsymbol{\lambda} := \begin{bmatrix}\Delta\boldsymbol{\lambda}_b \\ \Delta\boldsymbol{\lambda}_f\end{bmatrix} = -\boldsymbol{\Phi}\boldsymbol{A}\bar{\boldsymbol{I}}^{-1}\boldsymbol{f}_{act}, \tag{2.61}$$

where $\Delta\boldsymbol{\lambda} = \boldsymbol{\lambda} - \boldsymbol{\lambda}_{bias}$ denotes the contributions only due to the input forces, and $\Delta\boldsymbol{\lambda}_b \in \Re^{c_b \times 1}$, $\Delta\boldsymbol{\lambda}_f \in \Re^{c_f \times 1}$, denote the contributions to the contact constraints and friction constraints, respectively. The matrix $\boldsymbol{\Phi}$ can be decomposed into the four-block matrix

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_{11} & \boldsymbol{\Phi}_{12} \\ \boldsymbol{\Phi}_{12}^T & \boldsymbol{\Phi}_{22} \end{bmatrix} \tag{2.62}$$

to show more clearly the contribution to each set of constraints. The contribution from the inputs to each set of constraint forces can be expressed more clearly as

$$\begin{bmatrix} \Delta\boldsymbol{\lambda}_b \\ \Delta\boldsymbol{\lambda}_f \end{bmatrix} = -\begin{bmatrix} \boldsymbol{\Phi}_{11}\boldsymbol{A}_{b1}\bar{\boldsymbol{I}}_1^{-1} + \boldsymbol{\Phi}_{12}\boldsymbol{A}_{f1}\bar{\boldsymbol{I}}_1^{-1} \\ \boldsymbol{\Phi}_{12}^T\boldsymbol{A}_{b1}\bar{\boldsymbol{I}}_1^{-1} + \boldsymbol{\Phi}_{22}\boldsymbol{A}_{f1}\bar{\boldsymbol{I}}_1^{-1} \end{bmatrix} \boldsymbol{f}_{act1}, \tag{2.63}$$

which takes into account (A.8) and that the object is not directly actuated (2.54). An analytical expression of $\boldsymbol{\Phi}$ for the simplified model (2.47) is obtained in Section 3.1.

2.7.2 Acceleration of the system

Given an input $\boldsymbol{f}_{act}$, the final acceleration of the system can be obtained by substituting (2.58) into (2.54). The complete expressions for both the acceleration of the snake robot and object can be found in [25, 26]. In this paper we will find some simplifications relating the acceleration of the object $\boldsymbol{a}_{obj}$ as a function of the body constraints $\boldsymbol{\lambda}_b$. The twist $\boldsymbol{a}_{obj}$ encodes both linear and angular velocity of the object, as discussed in Appendix A. The complete acceleration of both systems can be written in vector form as

$$\boldsymbol{a} = \boldsymbol{a}_{act} + \boldsymbol{a}_{bias}, \tag{2.64}$$

where the terms $\boldsymbol{a}_{act}, \boldsymbol{a}_{bias} \in \Re^n$ defined as

$$\boldsymbol{a}_{act} := \frac{1}{m_s}\bar{\boldsymbol{I}}^{-1}\boldsymbol{\Phi}^{\perp}\boldsymbol{f}_{act}, \tag{2.65}$$

$$\boldsymbol{a}_{bias} := \frac{1}{m_s}\bar{\boldsymbol{I}}^{-1}\boldsymbol{\Phi}^{\perp}\left(\boldsymbol{f}_{ext} - \boldsymbol{p}\right) + \bar{\boldsymbol{I}}^{-1}\boldsymbol{A}^T\boldsymbol{\Phi}\dot{\boldsymbol{A}}\boldsymbol{v} \tag{2.66}$$

correspond to the contributions to the acceleration of the object due to the input forces and other terms, respectively. The new projector $\boldsymbol{\Phi}^{\perp}$ defined as

$$\boldsymbol{\Phi}^{\perp} := \left(\boldsymbol{1} - \boldsymbol{A}^T\boldsymbol{\Phi}\boldsymbol{A}\bar{\boldsymbol{I}}^{-1}\right) \tag{2.67}$$

is a projector from inputs to the space orthogonal to the constrained space $\mathcal{C}$ (c.f. Fig. 3.1(b)). Both projectors $\boldsymbol{\Phi}$ for the constrained subspace (2.58) and $\boldsymbol{\Phi}^{\perp}$ for the free subspace (2.64) do not depend explicitly on m_s or m_{obj}, but on their ratio κ. This simplifies the analysis of any snake robot contacting an object.

The acceleration of the snake robot $\ddot{\boldsymbol{q}}_s$ and object $\boldsymbol{a}_{obj}$ can be obtained analytically [25, 26]. However, in this paper we exploit the shape of $\boldsymbol{A}$ (c.f. (A.8)) and find a simplified expression for $\boldsymbol{a}_{obj}$

as a function of the input forces f_{act1}. This is due to the fact that the subtraction from the identity matrix in (2.67) only affects the diagonal terms of the matrix $A^T \Phi A \bar{I}^{-1}$. After all, the only input for the object is through the coupling with the snake robot, described by A_b.

First, assume the projector $\Phi^{\perp}$ is divided into four blocks as

$$\Phi^{\perp} = \begin{bmatrix} \Phi^{\perp}_{11} & \Phi^{\perp}_{12} \\ \Phi^{\perp}_{21} & \Phi^{\perp}_{22} \end{bmatrix}. \tag{2.68}$$

Since there are no inputs f_{act2} on the system, the terms $\Phi^{\perp}_{12}$ and $\Phi^{\perp}_{22}$ make no contribution to the acceleration of the system. The term $\Phi^{\perp}_{21}$, however, describes the mapping from the input forces of the snake robot $f_{act1} = B\tau_{act}$ to the acceleration of the object, and can be defined analytically as

$$\Phi^{\perp}_{21} := -\bar{I}_2^{-1} A_{b2}^T \left(\Phi_{11} A_{b1} \bar{I}_1^{-1} + \Phi_{12} A_{f1} \bar{I}_1^{-1} \right) \tag{2.69}$$

Taking into account the definition of $\Delta\lambda_b$ in (2.63), the acceleration of the object can be defined as

$$a_{obj} = \frac{1}{m_s \kappa} \bar{I}_2^{-1} A_{b2}^T \Delta\lambda_b + a_{bias,obj}, \tag{2.70}$$

where $a_{bias,obj} \in \Re^{n_{op}}$ is the bias acceleration for the object (this term will not be analyzed any further, since it is a term dependent on the state of the system).

The expression (2.70) is an improved version over the ones presented in [25, 26], since it isolates the terms from A that actually contribute to the acceleration of the object. In particular, the first term of (2.70) represents the contribution from the input to the snake robot $f_{act1} = B\tau_{act}$ to the acceleration of the object, through the coupling $\Delta\lambda$.

The (squared) norm of the acceleration of the object can be defined as

$$||a_{obj}||^2 := \langle a_{obj}, a_{obj} \rangle = a_{obj}^T I_{obj} a_{obj}, \tag{2.71}$$

which is a coordinate-independent metric (the operator $\langle \cdot, \cdot \rangle$ is the inner product induced by I [33, 48, 49]), and the acceleration of the object a_{obj} is defined in (2.70). This is not the usual *length* of the acceleration that is obtained with the Euclidean inner product, but rather a quantity that describes the power (strictly speaking, rate of change of power) of the acceleration of the object. It is a quantity that considers both linear and angular acceleration, while keeping consistency of units, thanks to the use of the metric tensor.

The solution of the system (i.e., obtaining accelerations and constraint forces as a function of the system's state and inputs) considering constraints can be solved as a Linear Complementary Problem (LCP) or reformulated as a Quadratic Program (QP). Presently, this is a standard procedure for modeling and simulation of multi-body systems. Due to simplicity's sake we do not add here the whole procedure, however there are several references. Specifically, we solve the QP as presented in [36] p. 224. Equivalently, solutions presented in [41] or more recently in [22] can be used.

2.7.3 Equations of Motion Rewritten

Previous analysis assumed a general model. For example, simplifications like $f_{act1} \equiv \tau_{act}$ and $f_{act1} \equiv \tau_{act}$ where made, as shown in (2.54). In the following we rewrite the equations for the accelerations of the snake robot and object in a more specific notation.

$$I a + p = f + A^T \lambda, \tag{2.72}$$

$$I = \begin{bmatrix} M_s & 0 \\ 0 & I_{obj} \end{bmatrix} \qquad a = \begin{bmatrix} \ddot{q}_s \\ a_{obj} \end{bmatrix}$$
$$p = \begin{bmatrix} h_s \\ p_{obj} \end{bmatrix} \qquad f = \begin{bmatrix} B\tau_{act} \\ f_{obj} \end{bmatrix}$$

Following the previous section, more general equations can be found for the acceleration of the snake robot and object as a function of the input joint torques. By solving for $\ddot{q}_s$ and a_{obj}, the motion of the snake robot and object can be obtained as

$$\ddot{q}_s = \left({}^{\ddot{q}_s}\Phi_{\tau_{act}}\right) \tau_{act} + \left({}^{\ddot{q}_s}\Phi_{f_{obj}}\right) f_{obj}, \tag{2.73}$$

$$a_{obj} = \left({}^{a_{obj}}\Phi_{\tau_{act}}\right) \tau_{act} + \left({}^{a_{obj}}\Phi_{f_{obj}}\right) f_{obj}, \tag{2.74}$$

where the auxiliary mappings ${}^{\ddot{q}_s}\Phi_{\tau_{act}} : \Re^{n_a} \to \Re^{n_s}$, ${}^{\ddot{q}_s}\Phi_{f_{obj}} : \Re^{n_{op}} \to \Re^{n_s}$, ${}^{a_{obj}}\Phi_{\tau_{act}} : \Re^{n_a} \to \Re^{n_{op}}$, and ${}^{a_{obj}}\Phi_{f_{obj}} : \Re^{n_{op}} \to \Re^{n_{op}}$ can be defined as

$${}^{\ddot{q}_s}\Phi_{\tau_{act}} := \frac{1}{m_s} \bar{M}_s^{-1} \left(1 - J_s^T \left(G_c^{-1}\right) J_s \bar{M}_s^{-1}\right) B \tag{2.75}$$

$${}^{\ddot{q}_s}\Phi_{f_{obj}} := \frac{1}{m_s \kappa} \bar{M}_s^{-1} J_s^T \left(G_c^{-1}\right) G^T \bar{I}_{obj}^{-1} \tag{2.76}$$

$${}^{a_{obj}}\Phi_{\tau_{act}} := \frac{1}{m_s \kappa} \bar{I}_{obj}^{-1} G \left(G_c^{-1}\right) J_s \bar{M}_s^{-1} B \tag{2.77}$$

$${}^{a_{obj}}\Phi_{f_{obj}} := \frac{1}{m_s \kappa} \bar{I}_{obj}^{-1} \left(1 - \frac{1}{\kappa} G \left(G_c^{-1}\right) G^T \bar{I}_{obj}^{-1}\right) \tag{2.78}$$

The (squared) length of the accelerations of the snake robot $||\vec{\ddot{q}}_s||^2$ or object $||\vec{a}_{obj}||^2$ can be obtained in an invariant way by considering its metric tensors, where the total expression can be divided into three terms as

$$||\ddot{q}_s||^2 = \tau_{act}^T \left(\Xi_{\tau_{act}}\right) \tau_{act} + f_{obj}^T \left(\Xi_{f_{obj}}\right) f_{obj} + \tau_{act}^T \left({}_{\tau_{act}}\Xi_{f_{obj}}\right) f_{obj} \tag{2.79}$$

$$||a_{obj}||^2 = \tau_{act}^T \left(\Omega_{\tau_{act}}\right) \tau_{act} + f_{obj}^T \left(\Omega_{f_{obj}}\right) f_{obj} + \tau_{act}^T \left({}_{\tau_{act}}\Omega_{f_{obj}}\right) f_{obj}. \tag{2.80}$$

We will concentrate on the contributions of the inputs of the snake robot. It can be verified that the auxiliary mappings $\boldsymbol{\Xi}_{\boldsymbol{\tau}_{act}}$ and $\boldsymbol{\Omega}_{\boldsymbol{\tau}_{act}}$, after some manipulation, can be defined as

$$\boldsymbol{\Xi}_{\boldsymbol{\tau}_{act}} := \frac{1}{m}\boldsymbol{B}^T \left(\mathbf{1} - \hat{\boldsymbol{J}}_s^T\right)^T \boldsymbol{M}_s^{-1} \left(\mathbf{1} - \hat{\boldsymbol{J}}_s^T\right) \boldsymbol{B} \tag{2.81}$$

$$\boldsymbol{\Omega}_{\boldsymbol{\tau}_{act}} := \frac{1}{\kappa m}\boldsymbol{B}^T \bar{\boldsymbol{M}}_s^{-1} \boldsymbol{J}_s^T \boldsymbol{G}_c^{-1} \boldsymbol{G}^T \bar{\boldsymbol{I}}_{obj}^{-1} \boldsymbol{G} \boldsymbol{G}_c^{-1} \boldsymbol{J}_s \bar{\boldsymbol{M}}_s^{-1} \boldsymbol{B} \tag{2.82}$$

where the auxiliary term

$$\hat{\boldsymbol{J}}_s^T := \boldsymbol{J}_s^T \boldsymbol{G}_c^{-1} \boldsymbol{J}_s \bar{\boldsymbol{M}}_s^{-1}$$

has been introduced for a more compact notation and any further simplification has been omitted for simplicity's sake. However, the linear relationship w.r.t. the masses of the system becomes evident.

2.7.4 Polar coordinates of the COM of the snake robot

In order to compare snake robots with different number of joints, it is necessary to parameterize the configuration of the robot with a set of parameters in common. The idea of using the polar coordinates of the of the snake robot $\boldsymbol{COM}_s$ w.r.t. the contact point with the object has been introduced in [25] and further explored in [26].

The parameterization of the COM of the snake robot $\boldsymbol{com}_s \in \Re^2$ can be defined as

$$\boldsymbol{com}_s := \{|COM_s|, \angle COM_s\}, \tag{2.83}$$

where the (unsigned) distance from the COM_s of the snake robot to the contact point is denoted by $|COM_s|$, and the angle between this vector and the link contacting the object is denoted as $\angle COM_s$. These quantities can be seen in Fig. 2.3(a), Fig. 3.1, and Fig. 3.2.

This allows us to parameterize any snake robot, regardless of the number of joints, in a consistent and unified manner, similar to how it has been done for robotic manipulators [28].

2.7.5 Summary

So far, the analysis of the coupling between the equations of the snake robot and the object have been fully obtained. These mappings usually ignore the dynamics of the object to be manipulated [38] or are conducted on a purely kinematic model [67]. However, for non-periodic motions where a steady-state is not to be assumed, these simplifications may not be enough. We consider that the analysis presented in this section can be used not only on snake robots, but also used as an improvement from [67, 68] for more general robotic systems.

If there were no passive wheels, the model would ignore the effects of $\boldsymbol{A}_{f1}$. In other words, the second block-column in (2.63) would vanish. However, if there are passive wheels attached to the snake robot, these terms can be included on the analysis of the system.

At the joint-space level we are talking about *configurations* of the robot (i.e., its generalized coordinates). However, several configurations will have similar inertial properties. We then, can propose the following definitions:

Definition 5. *Optimal Configurations:* Set of generalized coordinates $\boldsymbol{q} \in \Re^{n_s}$ of the snake robot that maximize the acceleration of the object, given the same input.
Definition 6. *Optimal Postures:* Set of postures of the snake robot, parametrized by the position of the COM of the robot as defined in (2.83), which maximize the acceleration of the object, given the same input.

Several configurations can produce postures with the same or similar inertial properties, since the mapping $\boldsymbol{q}_s \rightarrow \boldsymbol{com}_s$ is not injective (it is many-to-one). If the posture of the robot indeed has significant effects on the acceleration of the object, then snake robots with different number of joints could be compared, and better strategies for control could be formulated. This is similar to how robotic arms have benefited from understanding of their operational space properties over joint-space properties [28]. If the posture of the snake robot does not have a significant correlation to the acceleration of the object, then it would not be possible to find simple rules for optimal postures. In Chapter 3 we will show that indeed, the posture of the snake robot has a significant effect, regardless of the mass of the object to be manipulated. Furthermore, it will be shown that the optimal postures are independent of the additional frictional constraints due to passive wheels.

2.8 Slippage Ratio

2.8.1 Definition of Slippage Ratio

As stated in Chapter 1, it is an important problem to predict motion and not only forces, in order to understand the interaction between the snake robot and object and try to accomplish a task. If the task is to manipulate an object then it is desirable to maximize the motion of the object $||\boldsymbol{a}_{obj}||^2$ while minimizing the slippage of the snake robot $||\ddot{\boldsymbol{q}}_s||^2$. On the other hand, a snake robot could locomote using the environment as a source of propulsive forces or as a support, similar to the idea of climbing [53, 54, 55]. This case resembles more a walking robot where the contact with the environment is necessary for the robot to move. To the best of our knowledge, this distinction has not been studied with snake robots. To analyze this we propose the ratio of accelerations

$$sr := \frac{||\boldsymbol{a}_{obj}||^2}{||\boldsymbol{a}_{obj}||^2 + ||\ddot{\boldsymbol{q}}_s||^2}, \tag{2.84}$$

and call it *slippage ratio* which is a dimensionless scalar quantity bounded as $sr \in [0, 1]$. Using this ratio we can analyze the following three general situations:

- $sr \to 1$ which implies that the acceleration of the snake robot is minimal or that the object's acceleration is much greater than the robot's:
  $||\ddot{\boldsymbol{q}}_s||^2 \ll ||\boldsymbol{a}_{obj}||^2$ or $||\ddot{\boldsymbol{q}}_s||^2 \approx 0$.
- $sr \approx 0.5$ which implies a similar magnitude of acceleration for the two subsystems:
  $||\ddot{\boldsymbol{q}}_s||^2 \approx ||\boldsymbol{a}_{obj}||^2$.
- $sr \to 0$ which implies that the magnitude of the acceleration of the object is minimal:
  $||\ddot{\boldsymbol{q}}_s||^2 \gg ||\boldsymbol{a}_{obj}||^2$ or $||\boldsymbol{a}_{obj}||^2 \approx 0$.

This quantity can be seen as the ratio between a *desired* output and the total output. By analyzing the slippage ratio sr, given a configuration and input, we can understand better the behavior of the system.

Chapter 3

Optimal Configurations and Optimal Postures

In this Chapter, both a complex and simplified models of a snake robot contacting an object and the environment will be analyzed. It will be shown that the posture of the snake robot, parameterized by (2.83) has a significant impact on the object's acceleration. At the same time, it will be shown that the instantaneous solutions to the system, compatible with all the non-penetration and friction constraints, are not heavily influenced by friction.

3.1 Simplified Interaction Between the Snake Robot and an Object

In this section, a model that simplifies the snake robot contacting an object as two composite-rigid bodies (CRBs) is presented, denoted as *simple* model (c.f. Fig. 3.1). The full model considering the real equations of motion (2.54), denoted as *complex* model, will be used as a basis for comparison.

Although the complex model is generally used for the design of control laws and the analysis of the system, it is not trivial to extract the effects of some parameters from it. To be able to give an analytical description of how the shape of the body influences the system, the simple model (as presented in Section 2.6) will be used to gain intuition on the problem. The snake robot will be represented by B_1 and the object as B_2, unless noted otherwise. The motivation is that, if the snake robot locks its joints it behaves like a CRB, which gives an upper bound on the behavior of the system. The full articulated robot will never have more inertial effects than the CRB model.

Let's assume that two bodies B_1 and B_2 are contacting each other at a point p_c, and the matrix $\boldsymbol{T}_b$ spans the constraint force. Body B_1 has an additional constraint due to friction (e.g.,by the addition of a passive wheel), spanned by the matrix $\boldsymbol{T}_f$, acting at the point p_f. It is be assumed that only B_1 has forces that can be controlled (i.e, inputs that would be the joint actuators) represented by the vector $\boldsymbol{f}_{act1}$. The gravity forces acting on the bodies are $\boldsymbol{f}_{g1}$ and $\boldsymbol{f}_{g2}$ for B_1 and B_2, respectively. The only forces acting on the object are the gravity effects and the constraint force exerted by the snake robot. This means that the vector of forces $\boldsymbol{f}$ acting on the system (except

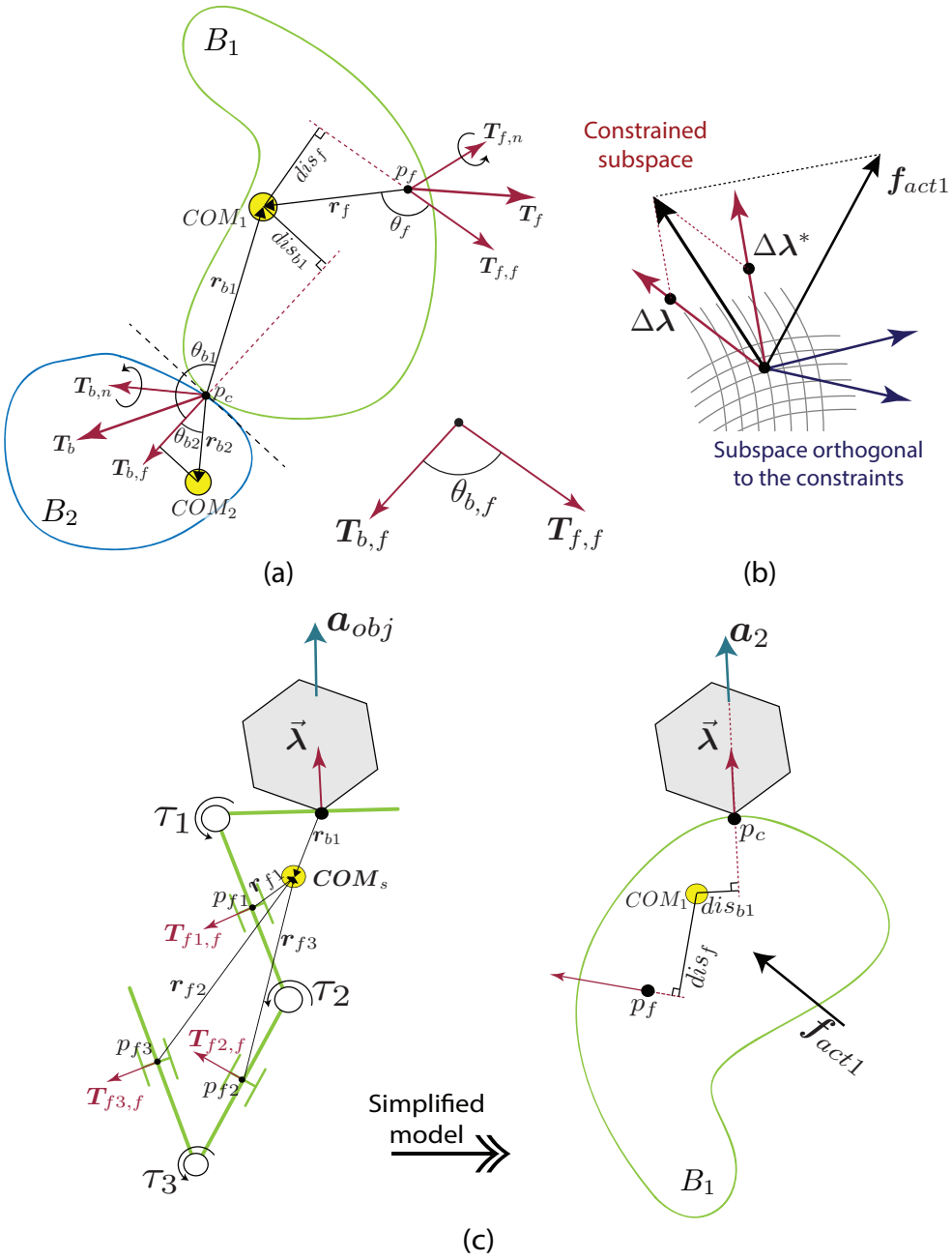


FIGURE 3.1: **Interaction between a snake robot and an object simplified** (a) Parameters of the COM of the bodies w.r.t. to the contact point and contact force $T_{b,f}$. (b) Projection of a input force into the constrained subspace. (c) Fitting the complex model of the snake robot into the simplified model

constraint forces) can be represented as

$$f := f_{act} + f_g, \qquad f_{act} := \begin{bmatrix} f_{act1} \\ \mathbf{0} \end{bmatrix}, \qquad f_g := \begin{bmatrix} f_{g1} \\ f_{g2} \end{bmatrix}.$$

3.1.1 Constraint forces

It is now possible to analyze the effect that the position of the COM of the bodies have on the constraint forces. By considering the definition of $\boldsymbol{A}$, its dependence on the matrices $\boldsymbol{T}_b$ and $\boldsymbol{T}_f$, and factorized inertia tensors $\bar{\boldsymbol{I}}_1$ and $\bar{\boldsymbol{I}}_2$, the system to be studied is

$$\Delta\boldsymbol{\lambda} = -\boldsymbol{\Phi}\begin{bmatrix} -\boldsymbol{T}_b^T\bar{\boldsymbol{I}}_1^{-1} \\ -\boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1} \end{bmatrix} \boldsymbol{f}_{act1}. \tag{3.1}$$

The matrix $\boldsymbol{\Phi} \in \Re^{c\times c}$

$$\begin{bmatrix} \boldsymbol{T}_b^T(\bar{\boldsymbol{I}}_1^{-1} + (1/\kappa)\bar{\boldsymbol{I}}_2^{-1})\boldsymbol{T}_b & \boldsymbol{T}_b^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f \\ \boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_b & \boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f \end{bmatrix}^{-1} \tag{3.2}$$

plays a fundamental role in describing the system since it has an influence on the magnitude of the constraint force exerted onto the object $\Delta\boldsymbol{\lambda}_b$, the magnitude of the friction forces $\Delta\boldsymbol{\lambda}_f$ and their coupling. In other words, it can be studied when and if the constraints due to friction have an impact on the wrench exerted on the object.

To analyze the matrix $\boldsymbol{\Phi}$ it is necessary to obtain an analytical expression. Under the assumption that the constraints are linearly independent, the matrix $\boldsymbol{\Phi}$ is a positive definite matrix (also interpreted as the metric tensor of the constrained subspace [33]) and, if divided into four blocks (c.f. (2.62)), an analytical expression can be obtained as

$$\boldsymbol{\Phi}_{11} := \left(\boldsymbol{T}_b^T\bar{\boldsymbol{I}}_{1,2}^{-1}\boldsymbol{T}_b - \boldsymbol{T}_b^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f(\boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f)^{-1}\boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_b\right)^{-1} \tag{3.3}$$

$$\bar{\boldsymbol{I}}_{1,2} := \bar{\boldsymbol{I}}_1^{-1} + \frac{1}{\kappa}\bar{\boldsymbol{I}}_2^{-1} \tag{3.4}$$

$$\boldsymbol{\Phi}_{12} := \boldsymbol{\Phi}_{11}\bar{\boldsymbol{\Phi}}_{12} \tag{3.5}$$

$$\bar{\boldsymbol{\Phi}}_{12} := (\boldsymbol{T}_b^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f)(\boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f)^{-1} \tag{3.6}$$

$$\boldsymbol{\Phi}_{21} = \boldsymbol{\Phi}_{12}^T \tag{3.7}$$

$$\boldsymbol{\Phi}_{22} := (\boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f)^{-1} + \bar{\boldsymbol{\Phi}}_{12}^T\boldsymbol{\Phi}_{11}\bar{\boldsymbol{\Phi}}_{12}. \tag{3.8}$$

The terms $\boldsymbol{T}_b^T\bar{\boldsymbol{I}}_{1,2}^{-1}\boldsymbol{T}_b$ and $(\boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f)^{-1}$ are positive definite matrices that represent the magnitude of the wrenches $\boldsymbol{T}_b$ and $\boldsymbol{T}_f$, respectively, with the inertia parameters as a metric tensor [33, 25]. These terms never vanish. The term $\boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_b$ represents the coupling between the kinematic constraints and the friction constraints. This term can vanish under certain conditions making the constraints decoupled. In other words, it is necessary to further study the matrix $\boldsymbol{\Phi}$ in order to understand how the inertial parameters of the system affect the constraint forces and their coupling.

Before diving into a more rigorous analysis, it is beneficial to make some simplifications. So far, the analysis presented has not made any assumptions on the matrices $\boldsymbol{T}_b$ and $\boldsymbol{T}_f$. $\boldsymbol{T}_b$ and $\boldsymbol{T}_f$

represent wrenches (moments and linear forces) and can be decomposed as

$$\boldsymbol{T}_b := \begin{bmatrix} \boldsymbol{T}_{b,n} \\ \boldsymbol{T}_{b,f} \end{bmatrix} \qquad\qquad \boldsymbol{T}_f := \begin{bmatrix} \boldsymbol{T}_{f,n} \\ \boldsymbol{T}_{f,f} \end{bmatrix},$$

Where the terms $\boldsymbol{T}_{b,n}$ and $\boldsymbol{T}_{f,n}$ represent moments, and $\boldsymbol{T}_{b,f}$ and $\boldsymbol{T}_{f,f}$ represent forces for $\boldsymbol{T}_b$ and $\boldsymbol{T}_f$, respectively. In this paper, it is assumed that only forces (and no moments) can be exerted between the bodies. This is equivalent to a frictionless point contact or hard finger contact models [37]. In the same manner, the passive wheels are modeled as only being able to produce a force in the perpendicular direction to the wheel. Under these considerations, the terms $\boldsymbol{T}_{b,n}$ and $\boldsymbol{T}_{f,n}$ vanish, allowing only forces to be transmitted. In addition, all quantities in the terms $\boldsymbol{T}_b^T \bar{\boldsymbol{I}}_{1,2}^{-1} \boldsymbol{T}_b$, $\boldsymbol{T}_f^T \bar{\boldsymbol{I}}_1^{-1} \boldsymbol{T}_b$ and $\boldsymbol{T}_f^T \bar{\boldsymbol{I}}_1^{-1} \boldsymbol{T}_f$ have to be represented on the same reference frame. We express the wrenches $\boldsymbol{T}_b$ and $\boldsymbol{T}_f$ w.r.t. the COM of the bodies. The vector $\boldsymbol{r}_{b1}$ and $\boldsymbol{r}_{b2}$ describe the position from the contact point p_c to the COM of B_1 and the COM of B_2, respectively. The vector $\boldsymbol{r}_f$ describes the position from the contact point of the passive wheel with the ground to the COM of B_1. By taking into account the transformations presented in Appendix A.1, it can be verified that the following expressions can be obtained

$$\boldsymbol{T}_b^T \bar{\boldsymbol{I}}_{1,2}^{-1} \boldsymbol{T}_b = (1 + \frac{1}{\kappa}) \boldsymbol{T}_{b,f}^T \boldsymbol{T}_{b,f} + (\boldsymbol{r}_{b1}^{\times} \boldsymbol{T}_{b,f})^T \bar{\boldsymbol{I}}_{c1}^{-1} \boldsymbol{r}_{b1}^{\times} \boldsymbol{T}_{b,f} + \frac{1}{\kappa} (\boldsymbol{r}_{b2}^{\times} \boldsymbol{T}_{b,f})^T \bar{\boldsymbol{I}}_{c2}^{-1} \boldsymbol{r}_{b2}^{\times} \boldsymbol{T}_{b,f}, \tag{3.9}$$

$$\boldsymbol{T}_f^T \bar{\boldsymbol{I}}_1^{-1} \boldsymbol{T}_b = \boldsymbol{T}_{f,f}^T \boldsymbol{T}_{b,f} + (\boldsymbol{r}_f^{\times} \boldsymbol{T}_{f,f})^T \bar{\boldsymbol{I}}_{c1}^{-1} (\boldsymbol{r}_{b1}^{\times} \boldsymbol{T}_{b,f}), \tag{3.10}$$

$$\boldsymbol{T}_f^T \bar{\boldsymbol{I}}_1^{-1} \boldsymbol{T}_f = \boldsymbol{T}_{f,f}^T \boldsymbol{T}_{f,f} + (\boldsymbol{r}_f^{\times} \boldsymbol{T}_{f,f})^T \bar{\boldsymbol{I}}_{c1}^{-1} (\boldsymbol{r}_f^{\times} \boldsymbol{T}_{f,f}), \tag{3.11}$$

where $\bar{\boldsymbol{I}}_{c1}^{-1}$ and $\bar{\boldsymbol{I}}_{c2}^{-1}$ represent the rotational inertias of B_1 and B_2 w.r.t. their COM (assuming unitary mass).

This shows the contribution of the inertial parameters of the bodies to the constraint forces. The terms $\boldsymbol{T}_{b,f}^T \boldsymbol{T}_{b,f}$, $\boldsymbol{T}_{f,f}^T \boldsymbol{T}_{b,f}$, and $\boldsymbol{T}_{f,f}^T \boldsymbol{T}_{f,f}$ do not depend on the position of the constraints w.r.t. the COM. Furthermore, the term $\boldsymbol{T}_{f,f}^T \boldsymbol{T}_{b,f}$ vanishes when the constraints $\boldsymbol{T}_b$ and $\boldsymbol{T}_f$ are orthogonal, regardless of their position w.r.t. each other. However, this does not guarantee that the term $(\boldsymbol{r}_f^{\times} \boldsymbol{T}_{f,f})^T \bar{\boldsymbol{I}}_{c1}^{-1} (\boldsymbol{r}_b^{\times} \boldsymbol{T}_{b,f})$ vanishes. The second terms of both (3.10) and (3.11) are the moments contributed by the constraints and depend on their position and orientation w.r.t. the COM of the body.

With these expressions it is possible to explain the effects that the constraints have on the system, their relationship, and how do they affect each other. Furthermore, optimal configurations of the snake robot can be studied. Since the configuration of the robot change the position of its COM and its rotational inertia, the effects that $\bar{\boldsymbol{I}}_{c1}^1$, $\boldsymbol{r}_{b1}$, and $\boldsymbol{r}_f$ have to be investigated further.

3.1.2 Contribution of the robot's parameters to the contact force

The contribution mapping from $\boldsymbol{f}_{act1}$ to $\Delta\boldsymbol{\lambda}_b$ can be obtained by considering the quantity $\langle \Delta\boldsymbol{\lambda}, \Delta\boldsymbol{\lambda} \rangle$ which is a coordinate-independent measurement of the constraint forces. The metric tensor $\boldsymbol{G}_c$ induces a metric on the constrained subspace, and the magnitude of the constraint force can be

obtained as

$$\begin{aligned}\langle \Delta\boldsymbol{\lambda}\,, \Delta\boldsymbol{\lambda}\rangle &:= \frac{1}{m_1}(\Delta\boldsymbol{\lambda}^T)\bar{\boldsymbol{G}}_c(\Delta\boldsymbol{\lambda}) \\ &= \frac{1}{m_1}(\Delta\boldsymbol{\lambda}^T)(\Delta\boldsymbol{\lambda}^*),\end{aligned} \tag{3.12}$$

where the vector $\Delta\boldsymbol{\lambda}^* = (1/m_1)\bar{\boldsymbol{G}}_c\Delta\boldsymbol{\lambda}$ is the dual coordinates of $\Delta\boldsymbol{\lambda}$ (c.f. Fig. 3.1(b)). Then, the magnitude of each set of constraint forces can be obtained as

$$\langle \Delta\boldsymbol{\lambda}\,, \Delta\boldsymbol{\lambda}\rangle = \langle \Delta\boldsymbol{\lambda}_b\,, \Delta\boldsymbol{\lambda}_b^*\rangle + \langle \Delta\boldsymbol{\lambda}_f\,, \Delta\boldsymbol{\lambda}_f^*\rangle. \tag{3.13}$$

The quantity we are interested in is $\langle \Delta\boldsymbol{\lambda}_b\,, \Delta\boldsymbol{\lambda}_b^*\rangle$ as a function of $\boldsymbol{f}_{act1}$ and the parameters of the system. It is the only input the object B_2 has (c.f. (2.70)), since it is received through the contact from the snake robot B_1, through the contact force $\boldsymbol{\lambda}_b$. It can be verified that the next expression is obtained

$$\langle \Delta\boldsymbol{\lambda}_b\,, \Delta\boldsymbol{\lambda}_b^*\rangle := \boldsymbol{f}_{act1}^T\boldsymbol{\Psi}\boldsymbol{f}_{act1}, \tag{3.14}$$

$$\boldsymbol{\Psi} := \frac{1}{m_1}\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{A}_{b1}^T\boldsymbol{\Phi}_{11}(\boldsymbol{A}_{b1} + \bar{\boldsymbol{\Phi}}_{12}\boldsymbol{A}_{f1})\bar{\boldsymbol{I}}_1^{-1}. \tag{3.15}$$

The expression (3.14) is a quadratic term representing the contribution of the input forces, as a function of the system's parameters. We will focus our attention on the term $\boldsymbol{\Phi}_{11}$ which, since is a diagonal term of the (inverse) metric tensor $\boldsymbol{\Phi}$, has an important role on both the numerical stability of the system and in the mapping $\boldsymbol{f}_{act1} \to \Delta\boldsymbol{\lambda}$.

The matrix $\boldsymbol{\Phi}_{11}$ is composed of two elements

$$\boldsymbol{\Phi}_{11} := (a - b)^{-1}, \tag{3.16}$$

where the auxiliary terms

$$a := \boldsymbol{T}_b^T\bar{\boldsymbol{I}}_{1,2}^{-1}\boldsymbol{T}_b \tag{3.17}$$

$$b := \boldsymbol{T}_b^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f(\boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_f)^{-1}\boldsymbol{T}_f^T\bar{\boldsymbol{I}}_1^{-1}\boldsymbol{T}_b \tag{3.18}$$

are scalars under the assumption that the snake robot is contacting only one object. This allows us to quantify the norm of $\boldsymbol{\Phi}_{11}$ as

$$||\boldsymbol{\Phi}_{11}|| = |(a - b)^{-1}| = |a - b|^{-1} \geqslant ||a| - |b||^{-1}. \tag{3.19}$$

By inspecting (3.19), it is evident that the smaller the difference $|a| - |b|$, the stronger the transmitted contact forces. However, the system becomes numerically unstable as $|a| - |b| \to 0$, which can be interpreted as the matrix $\boldsymbol{G}_c$ loosing rank. This happens when the constraints become linearly dependent. Furthermore, the bigger the difference $|a| - |b|$ is, the smaller the contact force exerted onto the object B_2. The norm of the first term $|a|$ can be obtained from (3.9)

as

$$|a| = |\boldsymbol{T}_b^T \bar{\boldsymbol{I}}_{1,2}^{-1} \boldsymbol{T}_b| \leqslant a_1 + a_2 + a_3, \tag{3.20}$$

where the auxiliary terms

$$a_1 := |1 + \frac{1}{\kappa}|||\boldsymbol{T}_{b,f}||^2, \tag{3.21}$$

$$a_2 := ||\bar{\boldsymbol{I}}_{c1}^{-1}||^2||\boldsymbol{T}_{b,f}||^2(dis_{b_1})^2, \tag{3.22}$$

$$a_3 := |\frac{1}{\kappa}|||\bar{\boldsymbol{I}}_{c2}^{-1}||^2||\boldsymbol{T}_{b,f}||^2(dis_{b_2})^2, \tag{3.23}$$

have been defined for a more compact notation. The quantity $|b|$ can be obtained from (3.10) and (3.11) as

$$|b| = |\boldsymbol{T}_b^T \bar{\boldsymbol{I}}_1^{-1} \boldsymbol{T}_f (\boldsymbol{T}_f^T \bar{\boldsymbol{I}}_1^{-1} \boldsymbol{T}_f)^{-1} \boldsymbol{T}_f^T \bar{\boldsymbol{I}}_1^{-1} \boldsymbol{T}_b| \leqslant b_1, \tag{3.24}$$

$$b_1 = ||\boldsymbol{T}_{b,f}||^2 \frac{(|\cos(\theta_{b,f})| + ||\bar{\boldsymbol{I}}_c^{-1}||dis_f dis_{b1})^2}{1 + ||\bar{\boldsymbol{I}}_c^{-1}||dis_f^2}. \tag{3.25}$$

The terms dis_{b1} and dis_{b2} represent the (unsigned) distances from the COM of B_1 and B_2 to the line spanned by the contact force, normal to the contacting surfaces. The distance dis_f is the (unsigned) distance from the COM of B_1 to the line spanned by the friction force of the passive wheel with the ground. The angle $\theta_{b,f}$ is the angle between the constraint forces spanning vectors $\boldsymbol{T}_{b,f}$ and $\boldsymbol{T}_{f,f}$ (c.f. Fig. 3.1(a)). Simplifications regarding the cross product (as mentioned in Appendix A.1) have been exploited to arrive to these expressions (e.g., $||\boldsymbol{T}_{b,f}||^2||\boldsymbol{r}_{b1}||^2\sin^2(\theta_{b1}) = ||\boldsymbol{T}_{b,f}||^2(dis_{b1})^2$).

Then, a lower bound for $||\boldsymbol{\Phi}_{11}||$ as a function of the distances dis_{b1}, dis_{b2}, and dis_f can be proposed as:

$$\phi_{11}(dis_{b1}, dis_{b2}, dis_f; model) := ||\boldsymbol{\Phi}_{11}|| \tag{3.26}$$

$$\geqslant |a_1 + a_2 + a_3 - b_1|^{-1}, \tag{3.27}$$

where *model* represents the snake robot's parameters (e.g., inertial parameters like $\boldsymbol{I}_{c1}$).

The metric (3.26) is not intended to be a quantitative metric for the system, but a qualitative one. It describes the effect that the input forces will have on the constraint forces (and therefore, on the motion of the object) as a function of the parameters of the system, and more importantly, of the position of the COM of the snake robot B_1 w.r.t. the contact constraint location.

3.1.3 Fitting the complex model data to the simplified model

Since the simplified model makes the assumption that there is only one friction constraint, some simplifications must be made, since the real snake robot has several passive wheels. We assume that the contact point between the plane and the i-th passive wheel, p_{fi} for $i = 1, ..., n_w$, occur at the centroid of the link, so that the normal forces are known. $n_w < n_\ell$ denotes the number of links with wheels, where it is assumed that at least one doesn't have a passive wheel (c.f.

Fig. 2.4). The friction at the i-th wheel is spanned by $\boldsymbol{T}_{fi,f}$ in the direction perpendicular to the link. The quantities we are interested in are the vectors from to the COM of the snake robot $\boldsymbol{r}_{fi} := COM_s - p_{fi}$ and the angles $\theta_{fi} := \angle(\boldsymbol{T}_{fi,f}, \boldsymbol{r}_{fi})$. This gives an unsigned distance

$$dis_{fi} := ||\boldsymbol{r}_{fi}|||\sin(\theta_{fi})| \quad \text{for } i = 1, ..., n_w \tag{3.28}$$

The mean of the distances is taken and a *virtual* passive wheel is created with this data and used for the simplified model.

$$dis_f := \text{mean}(dis_{fi}) \tag{3.29}$$

The procedure can be seen in Fig. 3.1(c).

3.2 Scenarios considered

From existing literature on snake robots (or structurally similar hyper-redundant manipulators, tentacle arms, etc.) it is not possible to draw conclusions on how the contact with the environment (either through friction or rigid-body contacts as presented in Section 1.1) has a positive or negative effect on the system, or any effect whatsoever. This is a key difference of snake robots compared to robotic manipulators that try to avoid obstacles as much as possible.

To study this we can apply the framework proposed in this paper while changing the number and type of constraints and studying the resulting acceleration of the object. In general, we propose three different scenarios depending on the type of constraints present on the system as follows:

- Scenario 1: The snake robot is in contact with an object but unconstrained in any other way.
- Scenario 2: The snake robot is contacting one object and has passive wheels in all other links. The friction between the passive wheels and ground is bounded by its limit surface.
- Scenario 3: The snake robot is contacting one object and has passive wheels in all other links. The passive wheels impose (unbounded and bilateral) non-holonomic constraints.

Scenario 1 allows us to consider only the inertial properties of the system. It represents a system where friction between the snake robot and environment is so low that it could be neglected $\mu_s \to 0$. Scenario 2 on the other hand, allows us to study the effect that passive wheels have on the system, but with a limit depending on a friction coefficient $\mu_s > 0$. Scenario 3 considers ideal passive wheels and could be considered as the extreme case when $\mu_s \to \infty$, which is the most model used for studying undulatory locomotion with snake robots. It is then clear that any case of anisotropic friction will lie somewhere between Scenario 1 and Scenario 3. In all cases it is assumed that the COM of the object is aligned with $\boldsymbol{T}_{b,f}$. The different scenarios can be seen in Fig. 3.2(a)-(c).

The snake robot used for the calculations has the parameters described in Table 3.1. The angles of the joints are varied in the range $[-135°, 135°]$. The input to the system is the set of joint

TABLE 3.1: Parameters of the snake robot

Symbol	Value	Unit	Description
n	6		Number of DOFs of the system
n_a	3		Number of actuated joints
n_ℓ	4		Number of links
n_w	3		Number of links with passive wheels
m_i	1	[kg]	Mass of $link_i$, $i = 1, ..., n_\ell$
ℓ_i	0.15	[m]	Length of $link_i$, $1 = 1, ..., n_\ell$
$I_{com,i}$	0.002	[kg m^2]	Rotational inertia for the i-th link
$\boldsymbol{\tau}_{act} = [\tau_{a1}, \tau_{a2}, \tau_{a3}]^T$		[N m]	Input joint torques
μ_s	0.1		Coefficient of (static) friction used for Scenario 2

torques $\boldsymbol{\tau}_{act}$; its effect on the acceleration of the object will be studied over the quadratic region defined as

$$\bar{\tau} = \{\boldsymbol{\tau}_{act} : \boldsymbol{\tau}_{act}^T \boldsymbol{M}_s^{-1} \boldsymbol{\tau}_{act} \leqslant 1\}, \tag{3.30}$$

which is a meaningful region where the power is bounded (unlike the Euclidean ball $\boldsymbol{\tau}_{act}^T \boldsymbol{\tau}_{act}$). The norm of the acceleration (2.80) which is a scalar function $\Re_a^n \rightarrow \Re$ will be calculated over the input region (3.30) and the maximum value for each configuration will be extracted along the parameters of the snake robot (e.g., dis_{b1}) and compared for all three scenarios. Finally, a comparison with the simplified model will be shown. A summary of the procedure can be seen in Fig. 3.2(d).

3.3 Results: Best Postures for Pushing an Object

Fig. 3.3 shows the magnitude of the acceleration of the object $||\vec{\boldsymbol{a}}_{obj}||^2$ for the three proposed scenarios. Fig. 3.3(a) shows the acceleration of the object as a function of the position of the COM of the snake robot w.r.t. the contact point p_c. A line is fitted to the results for clarity. As it can be seen, there is a clear relationship between the posture of the snake robot and the acceleration of the object, regardless of the mass of the object. The object's mass is varied from $\kappa = 0.1$ to $\kappa = 100$. The closer the COM of the robot is to the line of action (i.e., $dis_{b1} \rightarrow 0$) the better. Fig. 3.3(b) and 3.3(c) show the same data but for Scenario 2 and 3, respectively. The passive wheels do increase the acceleration of the object, however, this increase is small. More importantly, the same trends can be seen in all scenarios, proving that the posture of the robot has a significant impact on the acceleration of the object, regardless of the additional friction forces. It is also interesting to see that although the friction coefficient is very small (c.f. Table 3.1), Scenario 2 behaves almost the same as Scenario 3 (ideal passive wheels). The maximum acceleration obtained for each scenario and mass ratio κ is extracted and compared in Fig. 3.3(d). The best $||\boldsymbol{a}_{obj}||^2$ is obtained when $\kappa = 1$ and it can be verified that the passive wheels only have a small contribution.

Good and bad postures can be analyzed then, based on the distance from the COM of the snake robot to the action line spanned by $\boldsymbol{T}_{b,f}$. Fig. 3.4(a) shows a *good posture* where the COM is almost

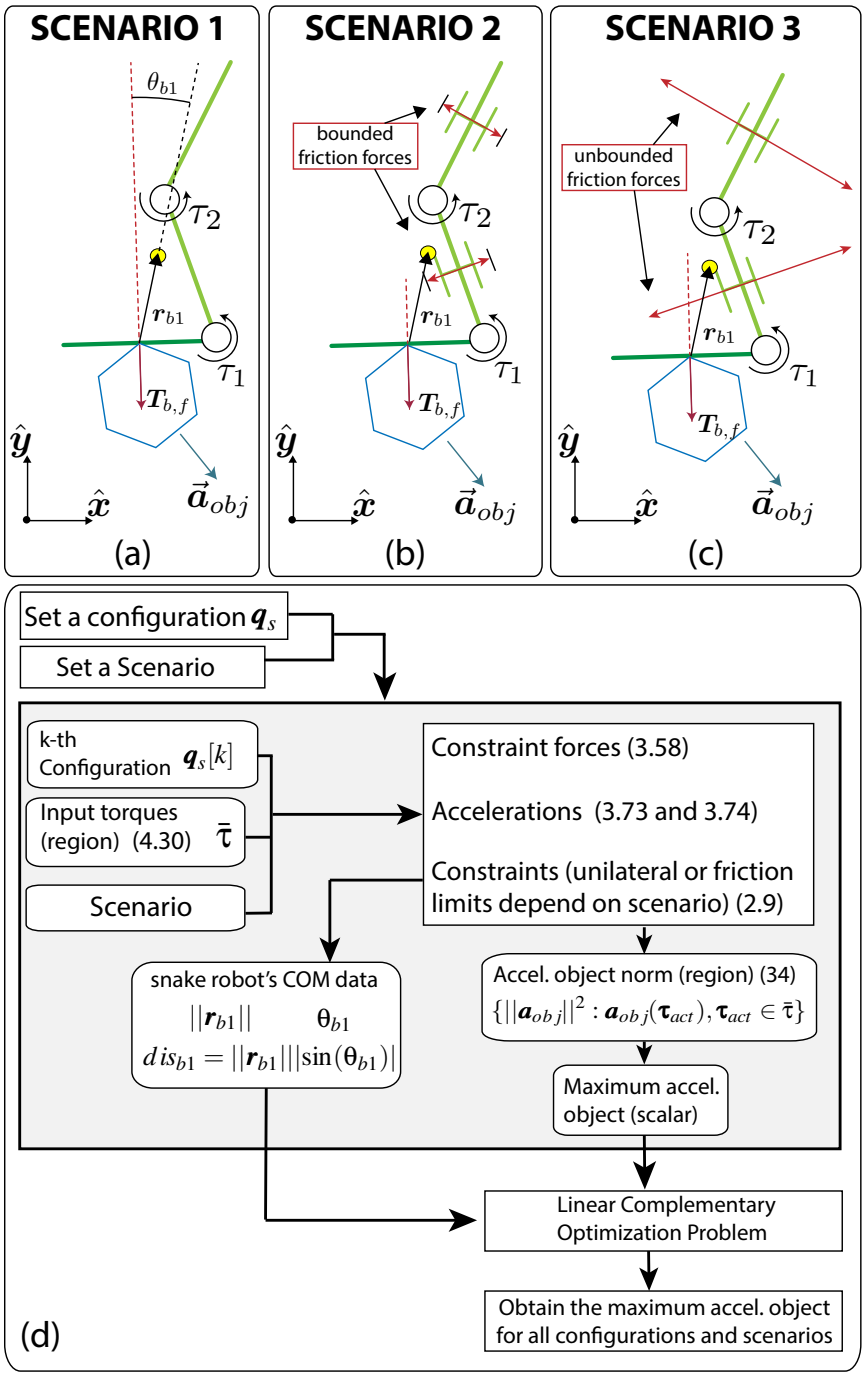


FIGURE 3.2: **Scenarios considered**
(a)-(c) Scenarios 1 through 3. (d) Summary of the procedure for obtaining the results. The used equations are shown if applicable

aligned with $T_{b,f}$ and a *bad posture* where the COM is almost perpendicular to $T_{b,f}$. Fig. 3.4(b) and 3.4(c) show the input region (3.30) for Scenario 1 and 3, respectively. The behavior is almost the same, although Scenario 3 shows a slightly bigger acceleration, an increase of 7.9%, actually. Fig. 3.4(d) and 3.4(e) show the input region (3.30) for Scenario 1 and 3, respectively, for the bad posture. As the scale shows, this postures shows a drastic decrease on the acceleration of the object, even though the input power is the same. Interestingly, the increase in the acceleration of Scenario 3 compared to Scenario 1 is more drastic. For this bad configuration Scenario 3

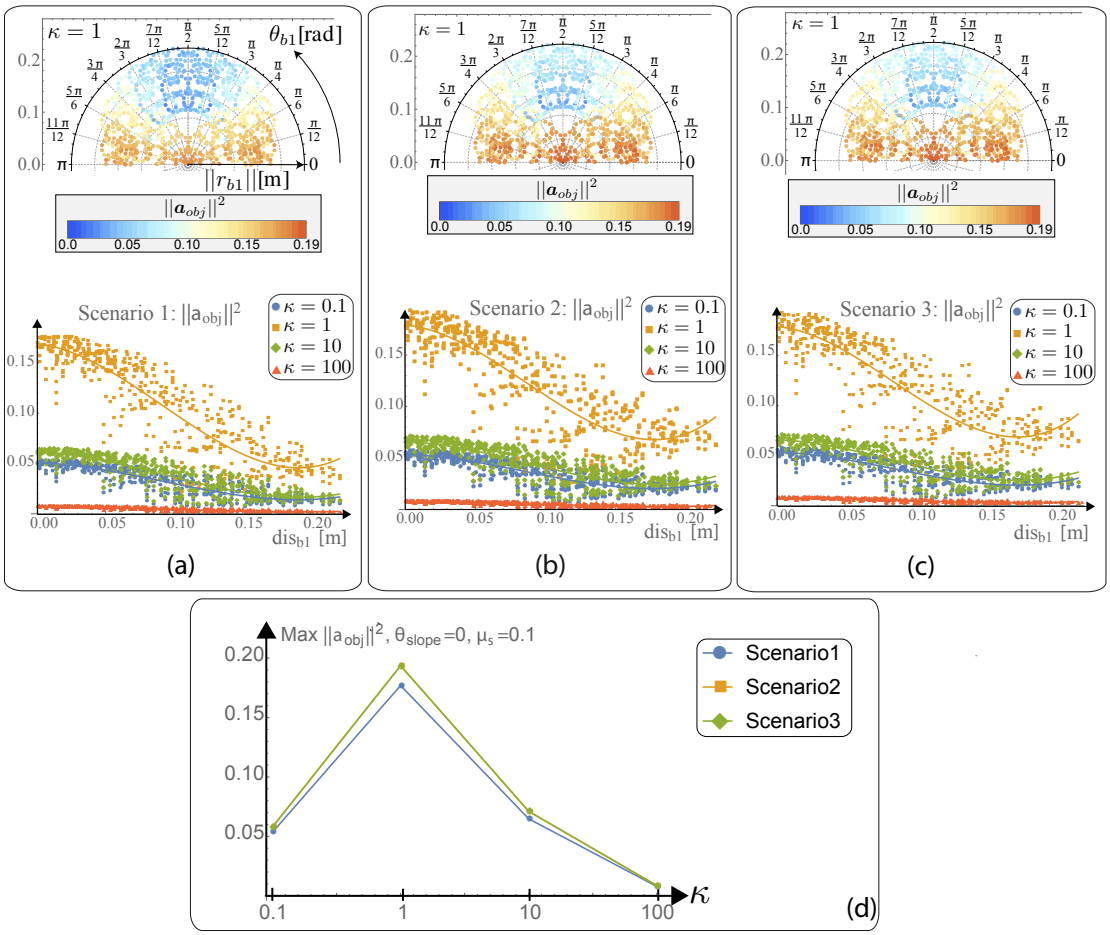


FIGURE 3.3: **Acceleration of the object**
Acceleration of the object as a function of the robot's posture parametrized by the position $||\boldsymbol{r}_{b1}||$ and angle θ_{b1} of its COM w.r.t. the contact with the object. Also shown as a function of $dis_{b1} = ||\boldsymbol{r}_{b1}|||\sin(\theta_{b1})|$. (a) Scenario 1. (b) Scenario 2. (c) Scenario 3. (d) Summary of the maximum object's acceleration as a function of the mass ratio κ

improved the acceleration of the object in 44.4%. Fig. 3.4(f) shows the increase on $||\boldsymbol{a}_{obj}||^2$ of Scenario 3 compared to Scenario 1. A negative value means that Scenario 1 was actually better. Results show a surprising behavior. Good postures benefit very poorly from the additional friction forces. However, bad postures show a significant improvement.

When $n_a > 3$ it is not possible to plot the input space in the same way as the previous examples (c.f. Figure 3.4). However, the configuration of the robot has similar effects regardless of the number of joints. A snake robot with two and four joints have been presented in [25] and [26], respectively.

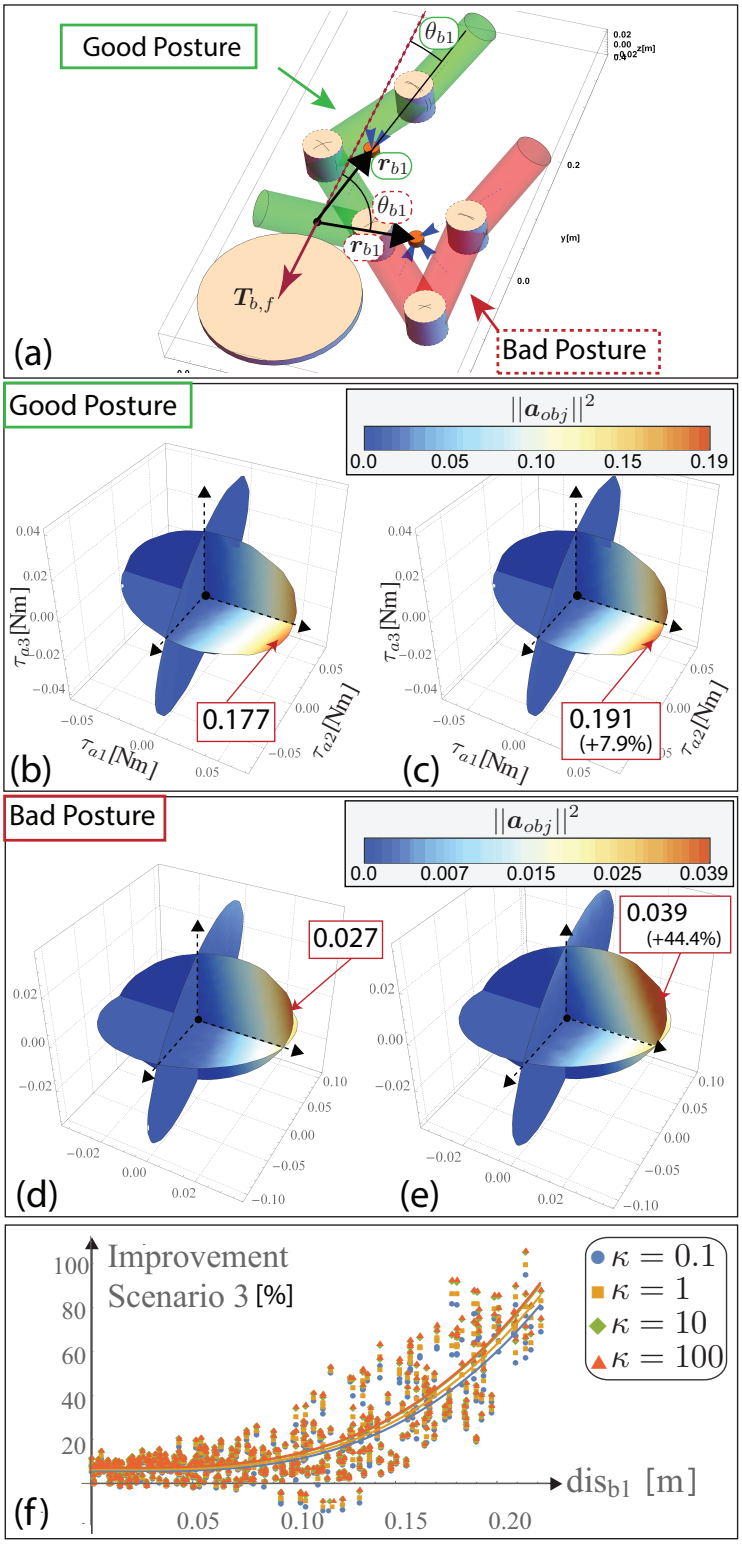


FIGURE 3.4: **Comparison Postures - Acceleration of the object**
(a) The snake robot with a good and bad posture (GP and BP, respectively). (b) Input space for Scenario 1 (GP). (c) Input space for Scenario 3 (GP). (d) Input space for Scenario 1 (BP). (e) Input space for Scenario 3 (BP). (f) Increase [%] for the $||a_{obj}||^2$ of Scenario 3 compared to Scenario 1

TABLE 3.2: Parameters of the Simulation for case study 1

Symbol	Value	Unit	Description
n	5		Number of DOFs of the system
n_a	2		Number of actuated joints
m_i	1	[kg]	Mass of $link_i$, $i = 1, ..., n_\ell$
ℓ_i	0.15	[m]	Length of $link_i$, $1 = 1, ..., n_\ell$
$I_{com,i}$	0.002	[kg m^2]	Rotational inertia for the i-th link
$\boldsymbol{\tau}_{act} = [\tau_{a1}, \tau_{a2}]^T$		[N m]	Input joint torques
μ_s	0.1		Coefficient of (static) friction used for Scenario 2

3.4 Results: Best Postures for Reducing Slippage

3.4.1 Case Study 1 - Snake robot with two joints

In order to show more specific and qualitative results, we apply the mappings and study a snake robot with two joints, the same as in the previous section. The small number of joints allows us to show graphically the magnitude of the studied norms as a function of the joint torques. First, we study a snake robot with the parameters described in Table 3.2. The snake robot is contacting an object with its tail (first link) and the contact occurs at the middle of the link (c.f. Figure 3.2). The angles of the joints are varied in the range $[-135^\circ, 135^\circ]$ every 10° (784 configurations in total) and the metrics (2.79), (2.80), and (2.84) are calculated within the quadratic region (3.30).

One example configuration can be seen in Fig. 3.5, where it is assumed that the object has a hundred times the mass of a link of the robot (i.e., $\kappa = 100$). The first, second, and third columns represent the three scenarios depicted in Fig. 3.2, respectively. A lighter color represents a higher value of the depicted norm. Fig. 3.5(a) shows the magnitude of the acceleration of the object $||\vec{\boldsymbol{a}}_{obj}||^2$. It can be seen that it barely changes regardless of the scenario (i.e., independently of the fact that the snake robot has or hasn't passive wheels, the object will accelerate the same given the same input). Fig. 3.5(b) shows the magnitude of the acceleration of snake robot $||\vec{\ddot{\boldsymbol{q}}}_s||^2$. This shows clearly that, even if the object's acceleration is similar for all three scenarios, the behavior of the snake robot changes. The addition of passive wheels (second and third columns) increase the area where the snake robot's slippage is minimal. Without passive wheels, the snake robot will slip in almost any direction of the input space.

The slippage ratio (2.84) gives quantitative information about the movement of the system and can be studied in the same way as the previous norms. Figure 3.5(c) shows the value of the slippage ratio for all three scenarios with several values for κ for one configuration. It can be seen that $sr \to 0$ in the region where there is no contact with the object (i.e., the snake robot can move freely and therefore $||\boldsymbol{a}_{obj}||^2 \to 0$). It is interesting to see that in all three scenarios it is always possible to make the object move. However, Scenario 2 (non-ideal passive wheels) has a limited region where the slippage ratio is high, compared to Scenario 3, where a whole region seems to give high values of slippage-ratio. These regions in the input space are highlighted in

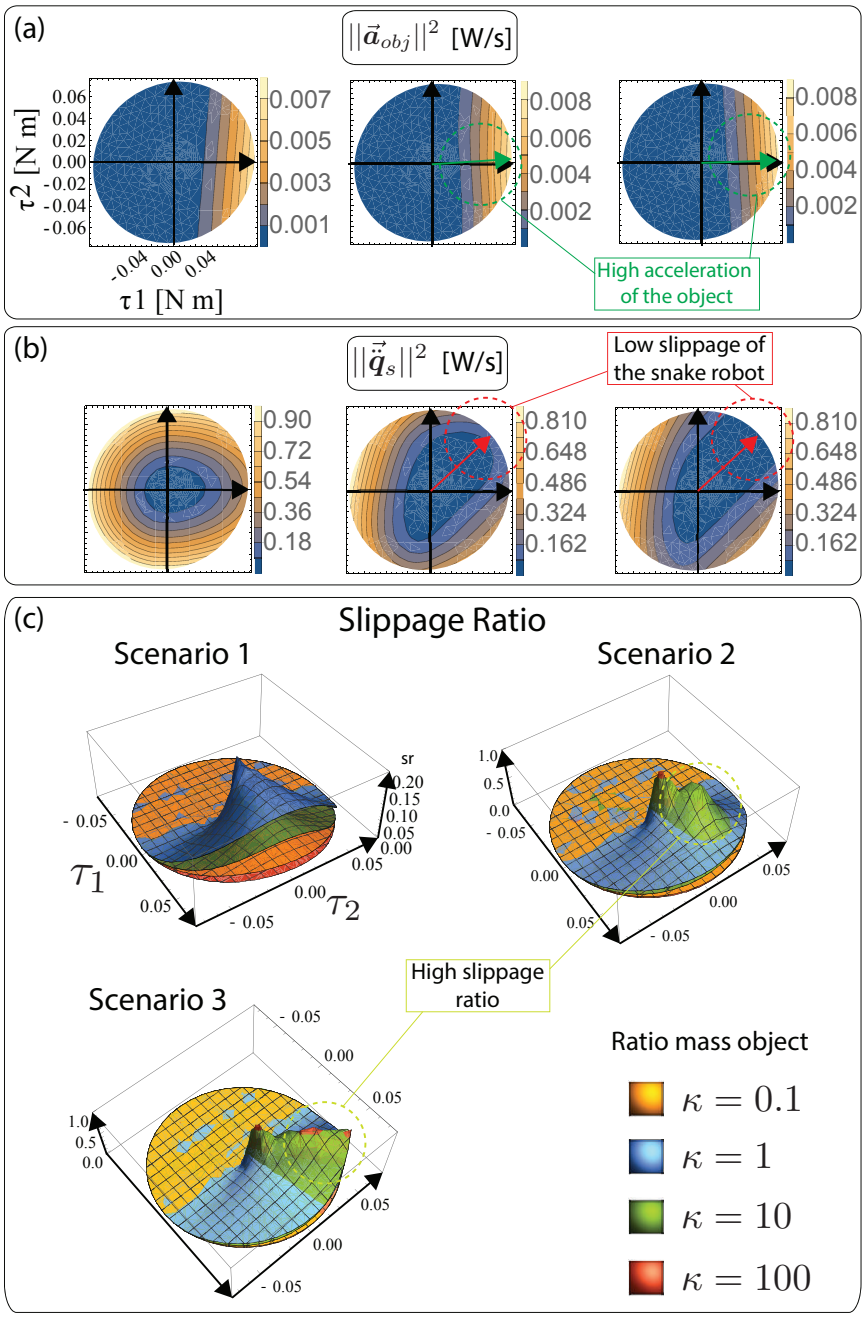


FIGURE 3.5: **Norms of motion of the system (two joints)**
The first, second, and third column represent scenario 1, scenario 2, and scenario 3, respectively. A higher value represents more power transmitted to the respective motion. The configuration of the robot is $\boldsymbol{q}_s = \{0, 0, 0, -135°, -135°\}$. (a) Acceleration of the object. (b) Acceleration of the snake robot. (c) Slippage ratio. Several values of κ are shown

Fig. 3.5(b) and 3.5(c). Regions where the slippage of the snake robot is minimized tend to have higher slippage ratio.

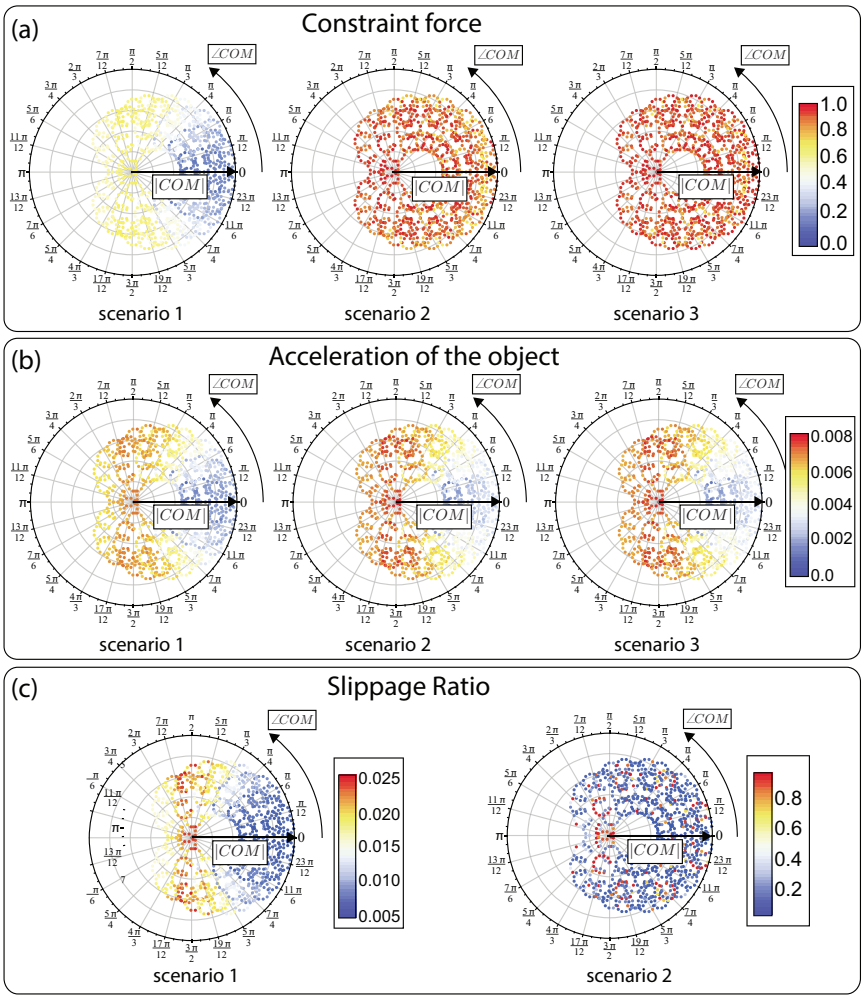


FIGURE 3.6: **Norms over all the configuration space (three joints)** Norms studied over all configurations of the snake robot. (a) Constraint force (from left to right: scenario 1, 2, and 3). (b) Acceleration of the object (from left to right: scenario 1, 2, and 3). (c) Slippage ratio (from left to right: scenario 1 and 2)

3.4.2 Case Study 2 - Snake robot with three joints

The proposed framework and metrics can be applied to a snake robot with any number of joints. In this section, a snake robot with three joints is studied. However, studying the three-dimensional input space could be cumbersome. Instead, the norms $||\boldsymbol{\lambda}||^2$, $||\boldsymbol{a}_{obj}||^2$ and slippage ratio (2.84) are studied as a function of the polar coordinates of the COM of the snake robot $(|COM|, \angle COM)$ w.r.t. the contact point, as discussed in previous sections.

Fig. 3.6(a) reports the result for the norm of constraint forces $||\boldsymbol{\lambda}||^2$. The polar plots show the results for scenario 1, 2, and 3, respectively. The higher the value, the bigger the constraint forces. It can be seen that in scenario 1 (negligible friction) there is a clear trend for configurations with the COM of the snake robot at angles 90° and -90° to have a higher impact on the wrench applied to the object. Although scenario 2, and 3 report a higher norm of the constraint forces, this is due to the addition of passive wheels. From this figure alone it is not possible to ascertain the impact on the object.

TABLE 3.3: Results of the norms

Concept	Scenario 1	Scenario 2	Scenario 3
Worst $\|\|\vec{a}_{obj}\|\|^2$	0.000763215	0.00107454	0.00107454
Best $\|\|\vec{a}_{obj}\|\|^2$	0.00785302	0.00861168	0.00861168
Worst sr	0.00138484	0.00185713	0.00185098
Best sr	0.0267348	0.988638	0.988643

Fig. 3.6(b) reports the result for the norm of the object's acceleration $||\boldsymbol{a}_{obj}||^2$. The polar plots report the results for scenario 1, 2, and 3, respectively. It can be seen that the addition of passive wheels (even ideal ones) have little impact on the acceleration of the object. However, the configuration of the snake robot, parametrized with the polar coordinates of its COM, have a clear and meaningful impact on the acceleration of the object.

Although basic intuition would tell that the addition of constraints (i.e., passive wheels) should have an impact on the force applied to the object (through the constraint forces $\boldsymbol{\lambda}$), and consequently on its acceleration $\boldsymbol{a}_{obj}$, this study shows that is not the case (at least, not that simply).

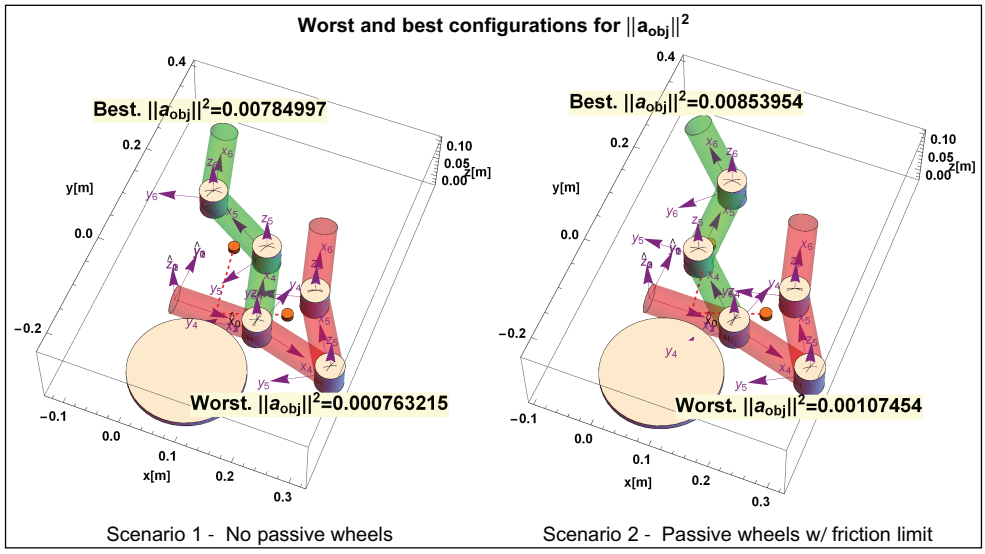
An important addition of this paper w.r.t. [25, 26] is the study of the slippage ratio sr. By studying the relationship between motions of both systems (snake robot and object) we can understand how the additional constraints have an impact on the system. A snake robot without passive wheels will slip as it pushes the object. Therefore, minimizing this motion while keeping a steady force on the object (and therefore producing an acceleration) is desirable. Fig. 3.6(c) shows the result of the slippage ratio (2.84). It can be seen that in Scenario 1 (without passive wheels) the same trend as with the object's acceleration appears. However, passive wheels (even non-ideal ones) have a big impact on the slippage ratio (take notice of the change of scale).

To show more clearly the impact of the configuration of the snake robot on the acceleration and slippage of the system, Fig. 3.7 shows the best and worst configurations for the acceleration of the object Fig. 3.7(a) and for slippage ratio Fig. 3.7(b). The results are summarized in Table 3.3. It can be seen that passive wheels (Scenario 2 and 3) have little impact on the acceleration, but a significant one on decreasing the slippage of the snake robot ($sr \to 1$).

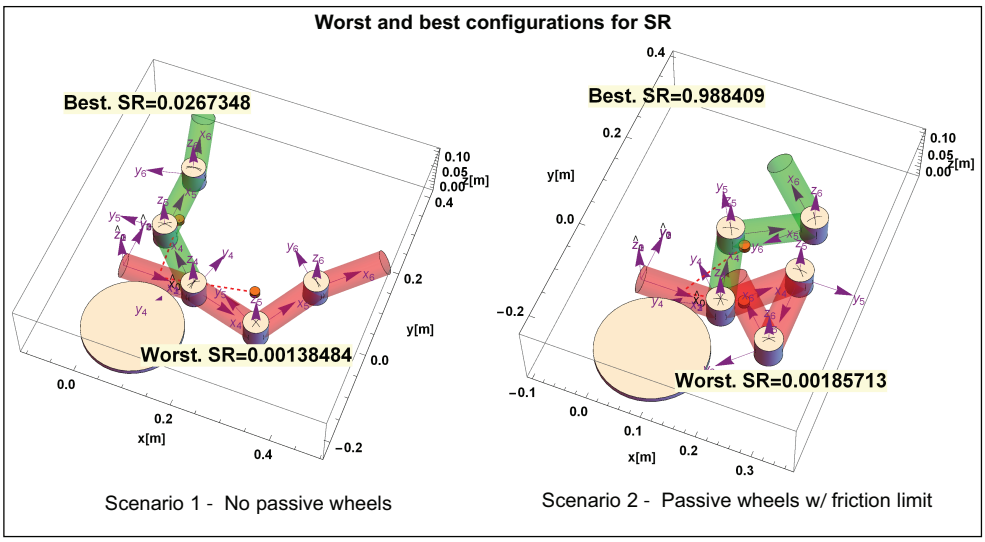
3.4.3 Case Study 3 - Snake robot with four joints

Finally, for the sake of showing that the results are consistent independently of the number of joints, a study case with a four-jointed snake robot is presented. The parameters of the snake robot are the same as in the previous sections. The results can be seen in Fig. 3.8.

The norm of the acceleration of the object can be seen in Fig. 3.8(b) for all three scenarios. It can be seen that, regardless of the friction with the ground, the results are the same. This shows that the analysis and model presented are valid for any snake robot, regardless of the number of joints.



(a)



(b)

FIGURE 3.7: **Representative configurations (three joints)**
Representative configurations chosen among the best and worst configurations of the snake robot.(a) Acceleration of the object (b) Slippage ratio

3.5 Results: Simplified Model vs. Complex Model

The simplified model, and the metric (3.26) can be compared to the behavior of the complex model presented in Section 3.3. The simplified model requires the parameters of the snake robot when simplified as a CRB. In particular, we require the rotational inertia of the snake robot w.r.t. its COM $||\boldsymbol{I}_{c1}||$. We propose to use the following two values

$$||\boldsymbol{I}_{c1}||_{min} := \min(||\boldsymbol{I}_{c1}||), \tag{3.31}$$

$$||\boldsymbol{I}_{c1}||_{max} := \max(||\boldsymbol{I}_{c1}||), \tag{3.32}$$

where $\min(||\boldsymbol{I}_{c1}||)$ and $\max(||\boldsymbol{I}_{c1}||)$ represent the minimum (e.g. the snake robot is curled up) and maximum (i.e. the snake robot is fully extended) rotational inertia that the snake robot

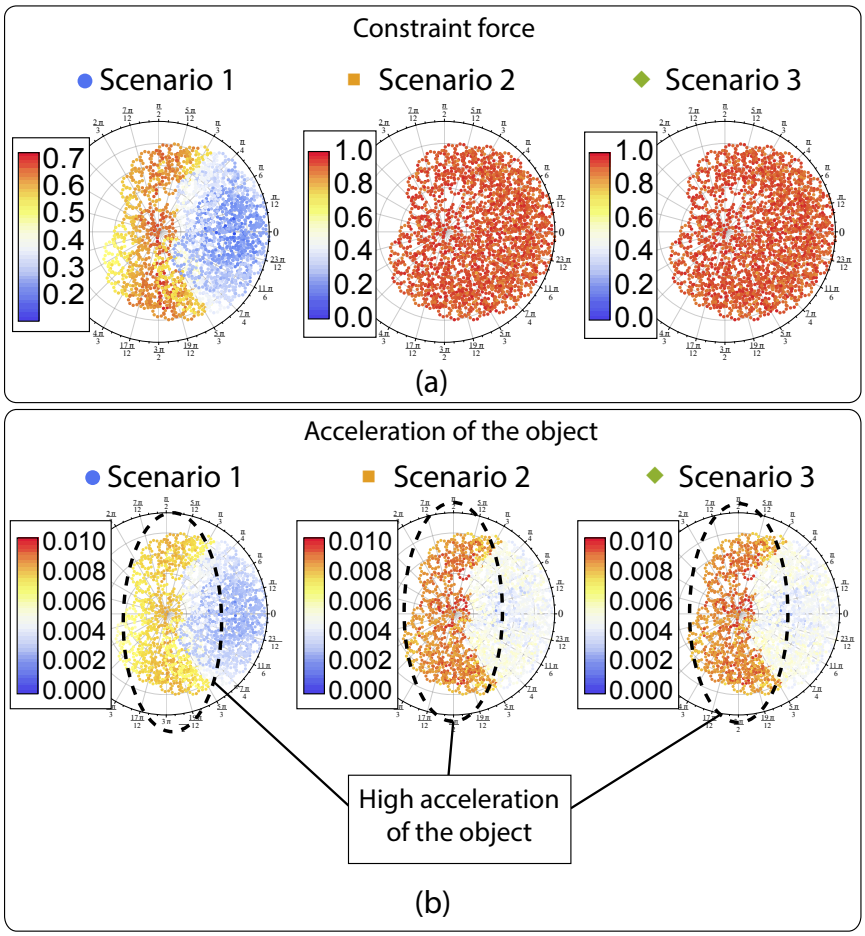


FIGURE 3.8: **Norms of motion of the system (four joints)**
The first, second, and third column represent scenario 1, scenario 2, and scenario 3, respectively. A higher value represents more power transmitted to the respective motion. The configuration of the robot is $\boldsymbol{q}_s = \{0, 0, 0, -135°, -135°\}$. (a) Constraint forces. (b) Acceleration of the object

presents.

Fig. 3.9 shows the bounds predicted by (3.26) when compared to the data obtained from the full model. In all cases Scenario 3 was considered, since the simplified model considers constraints due to friction without an explicit limit. Fig. 3.9(a) shows that the behavior is correctly predicted. The smaller dis_{b1} is, the bigger the acceleration of the object. The value $dis_f = 0$ was assumed. Fig. 3.9(b) shows the surface predicted by (3.26) when varying dis_{b1} and dis_f. The behavior is independent of dis_f, except for small values. This is a good match to the behavior of the complex model, where it has been shown that the extra friction forces have a small effect on the acceleration of the object.

It is clear that the trends described by the simplified model are similar to the ones of the full complex model. This analysis can help to provide some insight on the reason of this behaviour. However, it is important to notice that the simplified model has made several assumptions. This is due to the complex model of the snake robot, which is difficult and impractical to further analyze analytically. The simplified model is intended to provide qualitative conclusions regarding the behaviour of the system. It is not intended to be a rigorous quantitative metric.

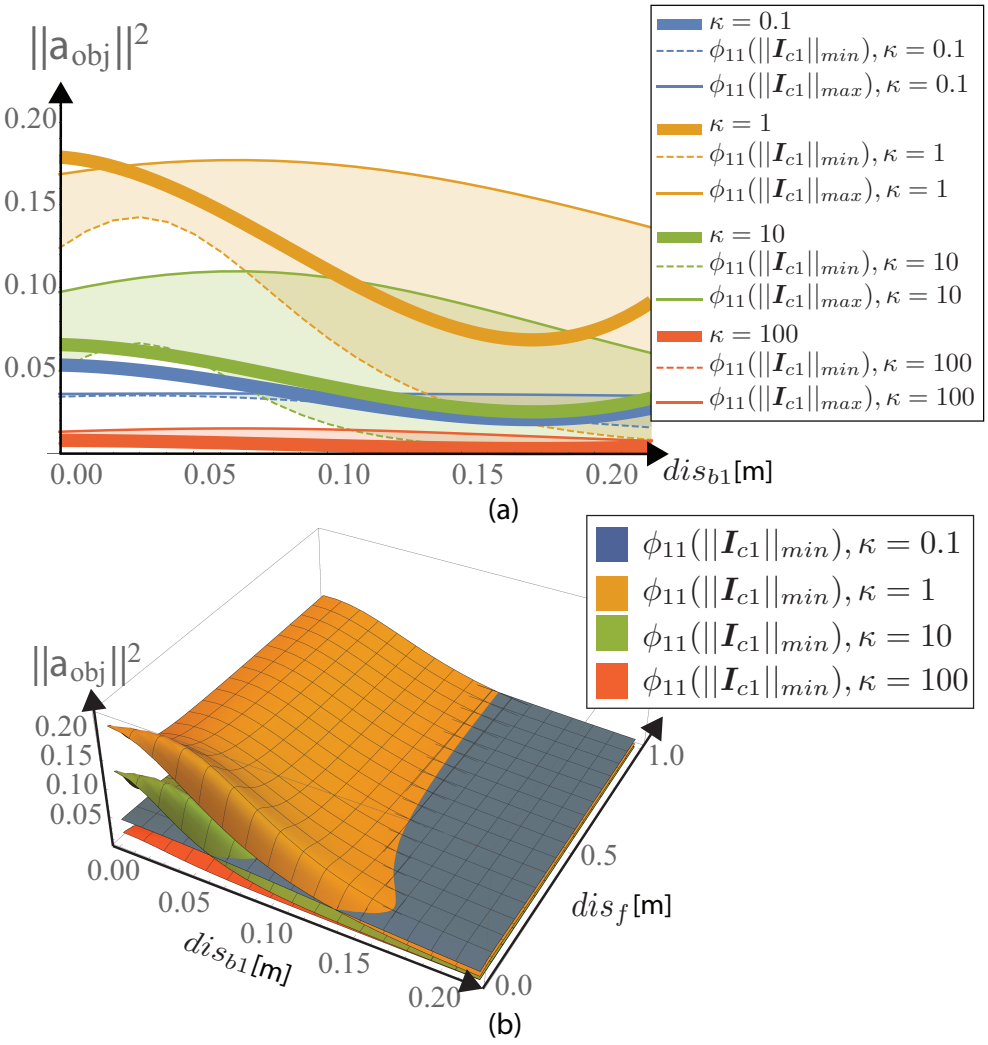


FIGURE 3.9: **Complex Model vs. Simplified Model**
Comparison between complex (Scenario 2) and simplified model. (a) Simplified model's bounds compared to the data collected from the complex model. (c) Surface predicted from the simplified model as a function of dis_{b1} and dis_f

Chapter 4

Experimental Results

In this Chapter we present a brief demonstration of how a snake robot can be used for pushing an object, while taking advantage of the results presented throughout the thesis. First, in Section 4.1 the experimental setup is fully explained. Section 4.2 presents a series of experiments where a snake robot pushes against an obstacle. The configuration and other parameters of the snake robot are measured. We propose a torque control law that maximizes the force imparted onto the object (and therefore its acceleration), while minimizing the slippage of the snake robot. This torque is obtained as the solution of an optimization problem described in Listing 4.1 and discussed in Section 4.1.2.

4.1 Experimental Setup

This section presents the details regarding the experimental setup The prototype is briefly explained in Section 4.1.1. The full details of the prototype are explained in Appendix B, Appendix C, and Appendix D.

4.1.1 Overview of the snake robot prototype

The snake robot is composed of three main parts: the body, the electronics, and the software. The overall design is shown in Fig. 4.1. The design of the body of the snake robot is more completely described in Appendix B. The body consists of eight links and seven joints (i.e., $n_a = 7$). The weight of each link is roughly 1 [kg]. Each joint is actuated by a servomotor. The snake robot is divided into two parts as discussed in Section 2.2: locomotion part and manipulation part (cf. Fig. 4.1).

To simulate an obstacle to be pushed, a 6DOF force sensor (DynPick 200R24) is used. This can be seen in 4.1(b). The sensor is guarded by a shell and fixed to the environment. The shell is free to rotate, since the obstacle is mounted on a bearing. This is to eliminate the effect of tangential friction between the snake robot's link an obstacle, something that is necessary to keep the assumptions discussed in Section 2.3 more realistic. This is similar [42], where the robot's shell was also free to rotate.

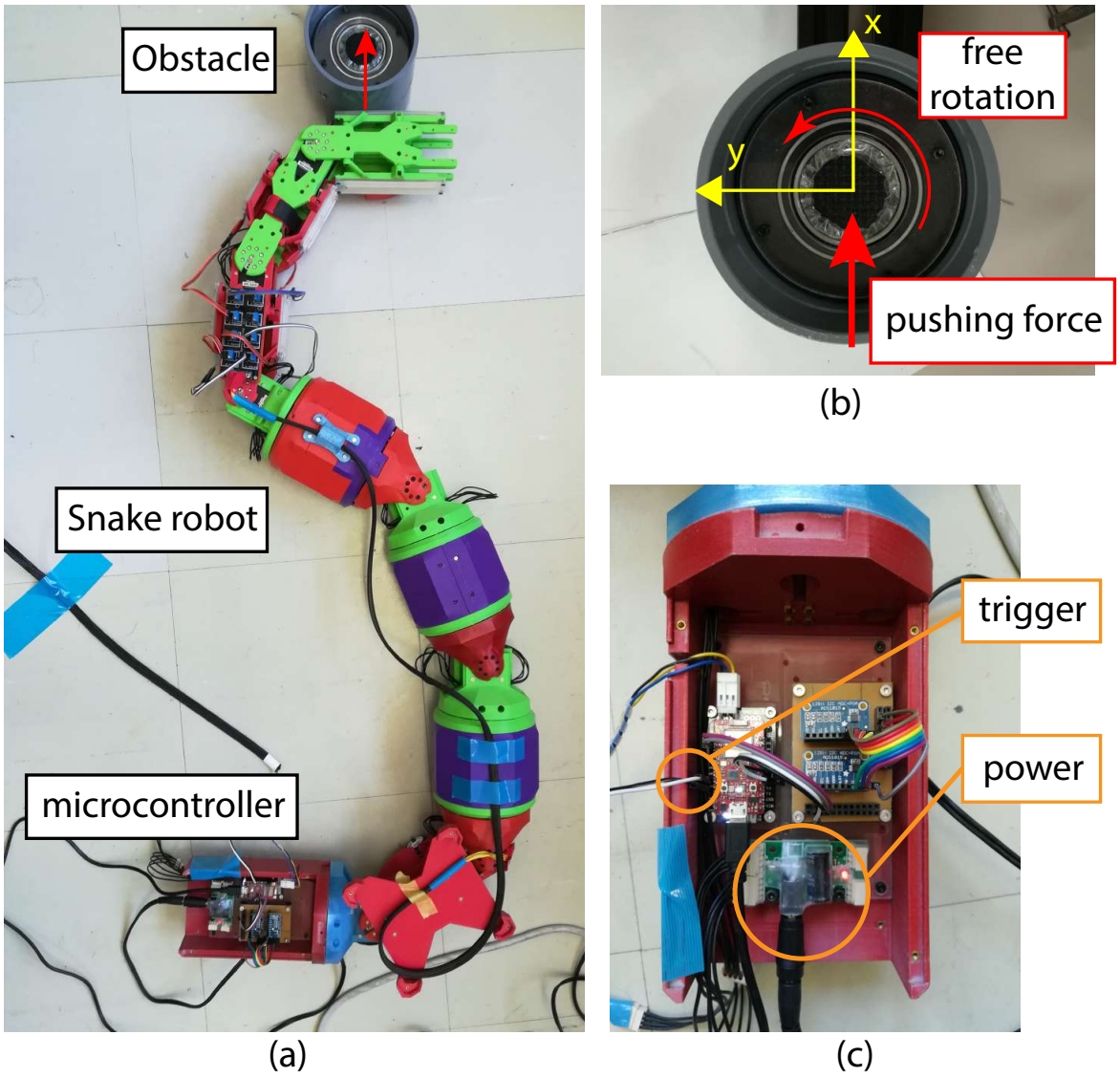


FIGURE 4.1: **Experimental Setup**
(a) The snake robot pushes against an object. (b) A force sensor reads the pushing force. (c) The tail of the robot controls the servomotors and send a signal to synchronize the force sensor readings and other feedback

The locomotion part's objective is to provide anisotropic friction force between the snake robot and the ground, enabling locomotion [43, 58]. The manipulation part, on the other hand has a spherical bearing on each link, in order to remove additional constraints. This has been researched previously in [24], which shows that if all links have the same frictional forces, it is very difficult to control the snake robot. Conceptually, removing these constraints is similar to what has been done in [47], where some links are slightly lifted from the ground to increase the manipulability of the snake robot. However, these may lead to a change in weight distribution which would be difficult to account for without the proper sensors. Therefore, the spherical bearings provide a similar behaviour without complicating the problem. The underside of the snake robot can be seen in Fig. 4.2.

The electronics of the prototype are described in Appendix C. A microcontroller on board of

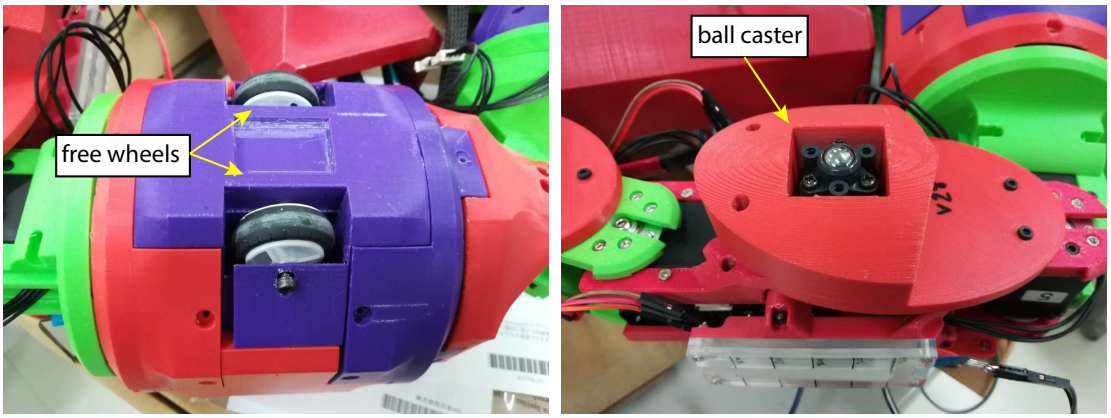


FIGURE 4.2: **Underside of the Snake Robot**
(a) Links corresponding to the locomotion part have passive wheels, in order to provide anisotropic friction. (b) Links corresponding to the manipulation part have a spherical bearing

the snake robot is in charge of all information gathering and low-level control. For example, the microcontroller communicates with the servomotors (either for sending a desired position/-command) or asking its current angle. Additionally, the microcontroller can perform basic timer-based routines, for example, perform gaits. The microcontroller can communicate via USB with a PC, where high-level control commands are computed.

4.1.2 Control Law

The objective is to obtain a vector of input torques $\boldsymbol{\tau}_{act}$ that maximizes the acceleration of the object to be pushed. However, as discussed in Section 2.8, it is possible to maximize the slippage ratio (2.84). By doing this a vector that maximizes the acceleration of the object, while minimizing the slippage of the robot, can be selected. We do this by proposing the problem as an optimization problem, as described in the following.

Listing 4.1: Optimization Problem: Maximize Slippage Ratio

```
Maximize the slippage ratio of the snake robot subject to the
    following constraints:
- Unilateral constraints: The snake robot can only push, not pull,
    the object
- Unitary input: The input must be within the unitary Euclidean
    ball
```

Mathematically speaking, the optimization problem can be stated as:

Optimization Problem I

$$\begin{aligned} \underset{\boldsymbol{\tau}_{act}}{\text{maximize}} &: sr \\ \text{s.t.} &: \Delta\boldsymbol{\lambda}_b \geqslant \mathbf{0} \\ &\boldsymbol{\tau}_{act}^T \boldsymbol{\tau}_{act} \leqslant 1 \\ \text{output} &: \{\boldsymbol{\tau}_{act}^*, \quad sr(\boldsymbol{\tau}_{act}^*)\} \end{aligned} \tag{4.1}$$

The first constraint $\Delta\boldsymbol{\lambda}_b \geqslant \mathbf{0}$ is to be interpreted element-wise. In other words, each constraint force corresponding to a body contact must be positive (i.e., pushing). The second constraint $\boldsymbol{\tau}_{act}^T \boldsymbol{\tau}_{act} \leqslant 1$ is a simple scalar constraint in order to have a bounded problem. The output of the optimization problem (4.1) are the values $\boldsymbol{\tau}_{act}^*$ which is the vector of input torques that maximizes the slippage ratio sr ($\boldsymbol{\tau}_{act}^*$ could also be interpreted as the argument that maximizes the optimization problem **argmax**) and the slippage ratio (cost function in the context of optimization) evaluated at $\boldsymbol{\tau}_{act}^*$.

By using the definition of slippage ratio (2.84) and definitions of accelerations (2.79) (2.80) and constraint forces (3.14) the optimization problem can be stated more clearly as

$$\begin{aligned} \underset{\boldsymbol{\tau}_{act}}{\text{maximize}} &: \frac{\boldsymbol{\tau}_{act}^T \boldsymbol{\Xi} \boldsymbol{\tau}_{act}}{\boldsymbol{\tau}_{act}^T (\boldsymbol{\Xi} + \boldsymbol{\Omega}) \boldsymbol{\tau}_{act}} \\ \text{s.t.} &: (\boldsymbol{\Phi}_{11} \boldsymbol{A}_{b1} \boldsymbol{M}_s^{-1} + \boldsymbol{\Phi}_{12} \boldsymbol{A}_{f1} \boldsymbol{M}_s^{-1}) \boldsymbol{\tau}_{act} \geqslant \mathbf{0} \\ &\boldsymbol{\tau}_{act}^T \boldsymbol{\tau}_{act} \leqslant 1 \\ \text{output} &: \{\boldsymbol{\tau}_{act}^*, \quad sr(\boldsymbol{\tau}_{act}^*)\} \end{aligned} \tag{4.2}$$

Unfortunately, because the input space is 7-dimensional, it cannot be plotted as in Section 3.3. Since the object is fixed in this case, we cannot maximize its acceleration. But since the acceleration of the object depends only on the snake robot (i.e., the object has no other means of propulsion), maximizing the force applied is equivalent. A flow diagram of the methodology used to control the snake robot can be seen in Fig. 4.3. The application starts with a setup routine that only needs to run once. Based on the parameters of the snake robot, a basic model is constructed. Then, the control loop starts. At every iteration the state of the robot is obtained. The angles $\boldsymbol{q}_s$ are measured and if outside of a desired limit, the robot stops to avoid damage. Then the forward kinematic mappings (e.g., the Jacobian's (2.1)) are obtained along the inertia matrix $\boldsymbol{M}_s$. Then the optimization problem (4.2) is solved, and the obtained torque $\boldsymbol{\tau}_{act}^*$ is sent to the servomotors. The history of the angles and torques, along a time stamp of the absolute time since the experiment began is saved as a CSV (Comma-separated Values) file, in order to reconstruct the scene off-line.

4.2 Optimal Postures - Experimental Results

Three experiments are performed to show an application of EAM. As discussed in Section 3.3, configurations where the COM is along the action line of pushing are optimal. However, the

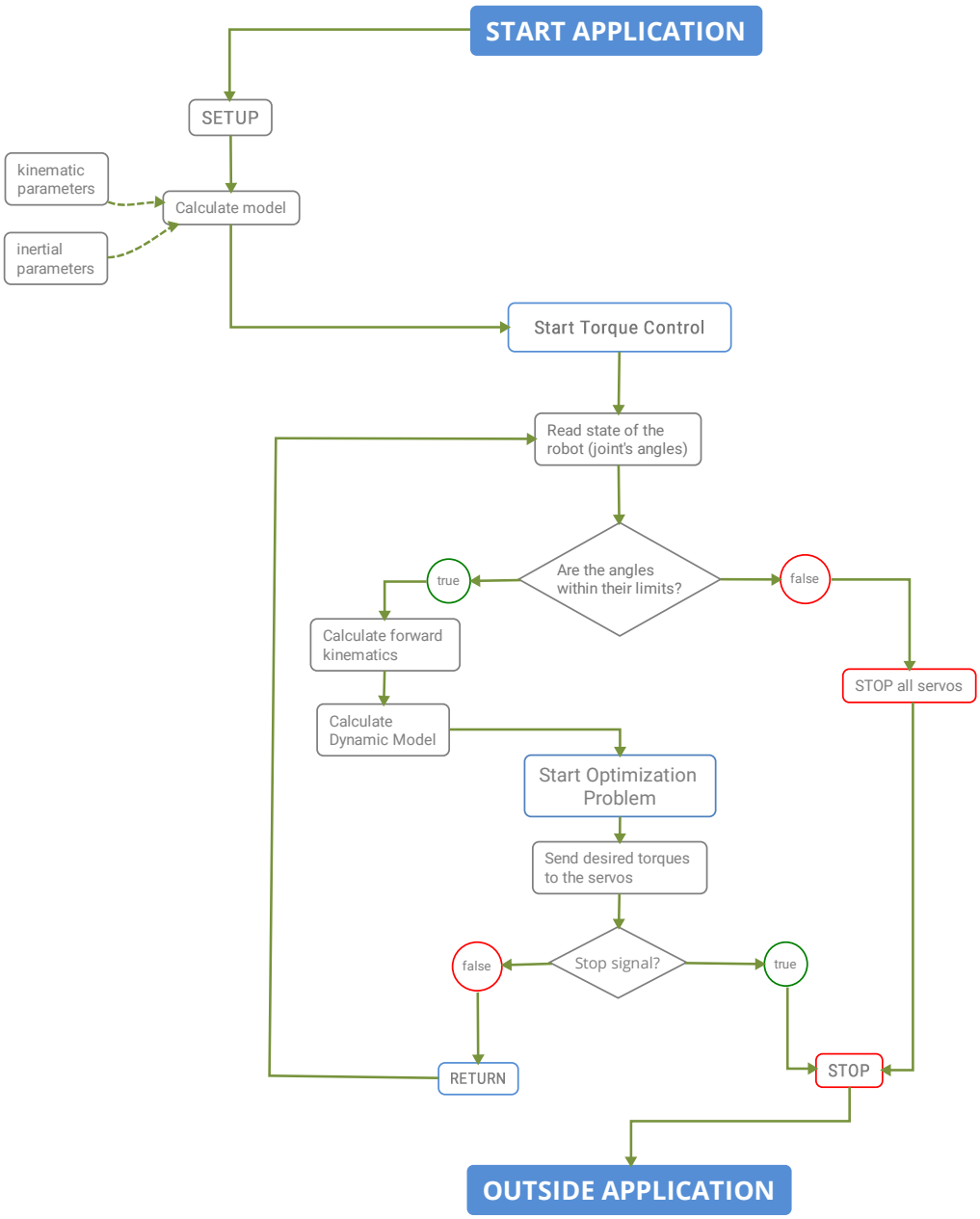


FIGURE 4.3: **Torque Control Flow Diagram**

analysis provided is only at one instant of time. In other words, it is very difficult to predict how the system will behave as the snake robot pushes. This is one problem of snake robots that are under-actuated.

The experimental setup is as shown in Fig. 4.1. There are three different setups to perform:

- Good Posture (pushing): In these trials, the snake robot is pushing the object *away* from the robot itself. By good posture we mean that the COM of the snake robot is close to the

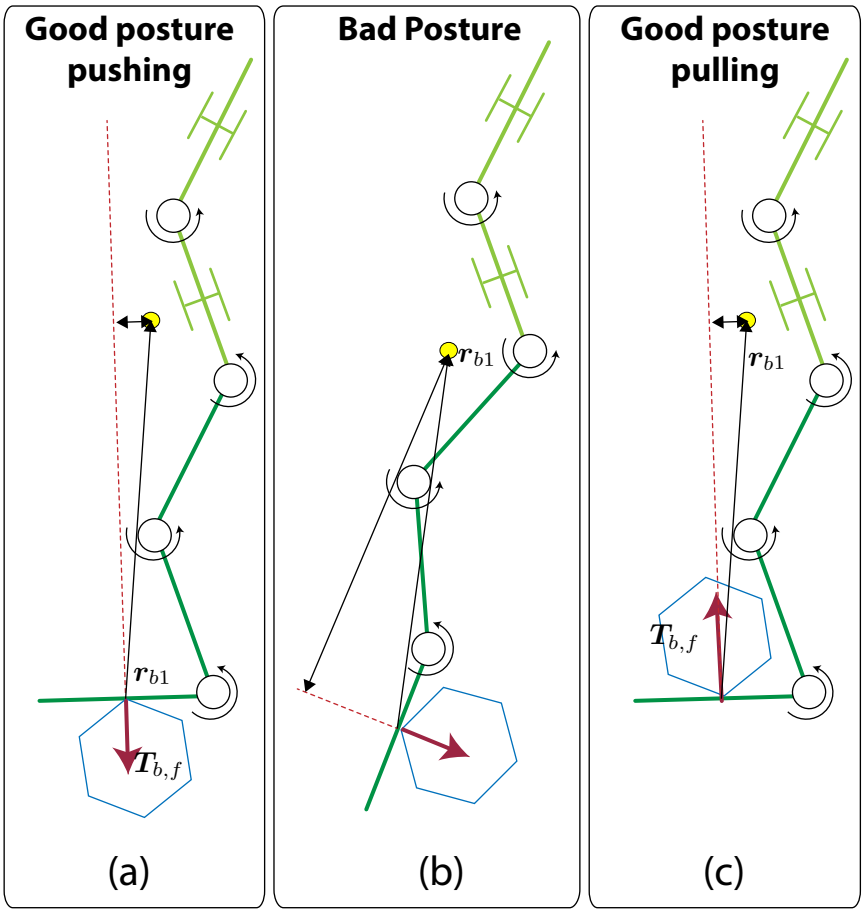


FIGURE 4.4: **Different types of setups for the snake robot experiment**
(a) A good posture pushing the object away. (b) A bad posture. (c) A good posture pulling the object

action line, as discussed in Section 3.3.

- Bad Posture: In these trials, the snake robot is pushing against the object, but the normal distance from the COM of the snake robot to the object is significantly bigger.
- Good Posture (pulling): In these trials, the snake robot is pulling the object *towards* the robot itself. The distinction between *pushing* or *pulling* is purely semantic, as it is still a positive non-penetration force.

The different setups can be seen in Fig. 4.4.

The snake robot is commanded to maximize the slippage ratio (2.84) at every iteration, given the information about the object's location (w.r.t. the robot), the parameters of the robot (e.g., kinematic and inertial parameters) and feedback from the joints' angles. Several trials are performed for each setup.

Fig. 4.5 shows the results for the first experimental setup: Good Posture (pushing). The time history of the COM's normal distance to the action line is shown in Fig. 4.5(a). Four trials were

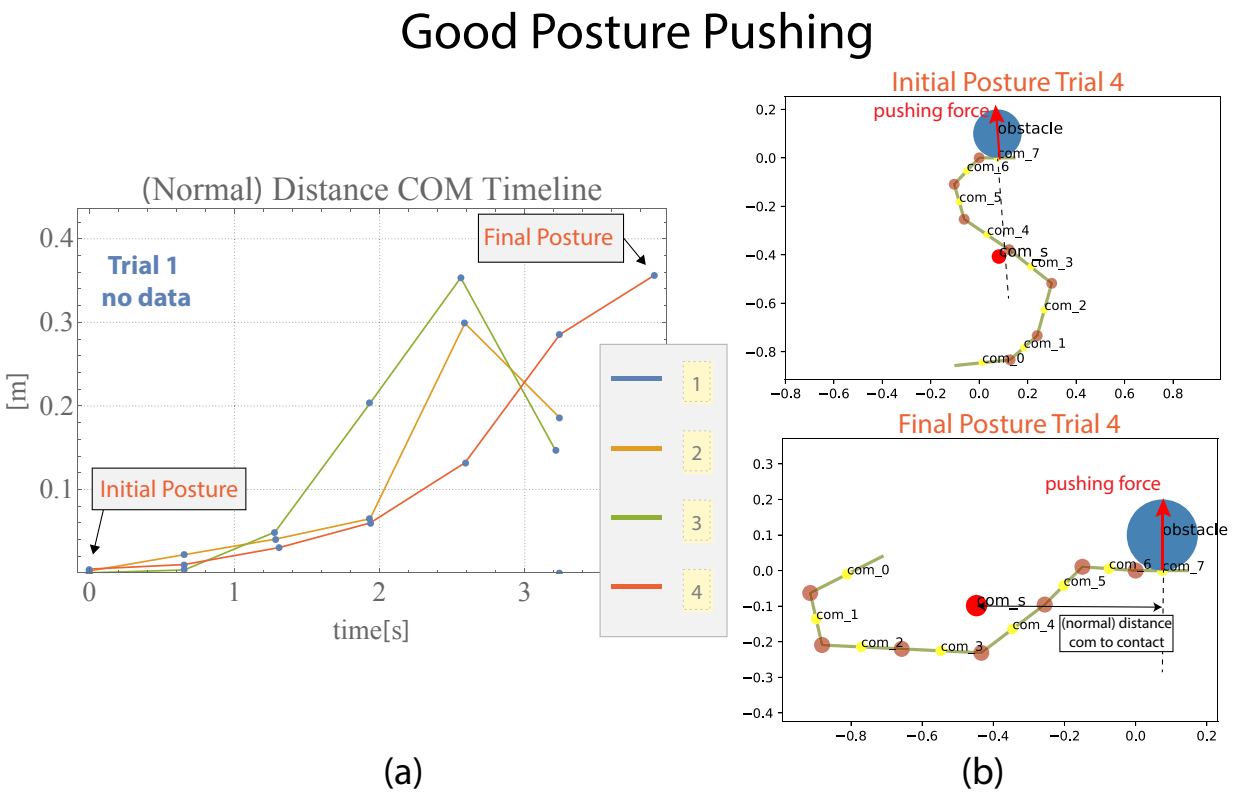


FIGURE 4.5: **Results experiment 1 - A snake robot with good posture pushing against an object**
(a) The time history of the normal distance from the COM of the snake robot to the pushing line for four trials (b) The initial and final robot's posture for the fourth trial

performed[1]. As it can be seen, when the robot pushes the obstacle away from itself, the COM tends to get further and further away. In other words, even if the robot starts at a good posture, as the system evolves in the time the robot tends to get into a bad configuration. 4.5(b) shows the robot's posture (reconstructed from the feedback history). As it can be seen, in all trials the COM starts almost perfectly aligned with the action line.

Fig. 4.6 shows the results for the second experimental setup: Bad Posture. The time history of the COM's normal distance to the action line is shown in Fig. 4.6(a). Three trials were performed. As it can be seen, the robot is not capable of moving much, but still the COM gets a little closer to the action line. In other words, the robot tends to *curl* itself. In the first trial, the snake robot lost contact very quickly and the experiment was terminated to avoid damage to the robot. 4.6(b) shows the robot's posture (reconstructed from the feedback history).

Fig. 4.7 shows the results for the third experimental setup: Good Posture while pulling. The time history of the COM's normal distance to the action line is shown in Fig. 4.7(a). Three trials were performed. The COM doesn't move away from the action line too much, in fact, it gets a little closer. In this case, it is very natural for the robot to curl itself. 4.7(b) shows the robot's posture (reconstructed from the feedback history).

[1]The CSV of the joints' angles for trial 1 was corrupted, so the system could not be reconstructed off-line. However the force sensor readings were saves successfully, as shown in Fig. 4.8

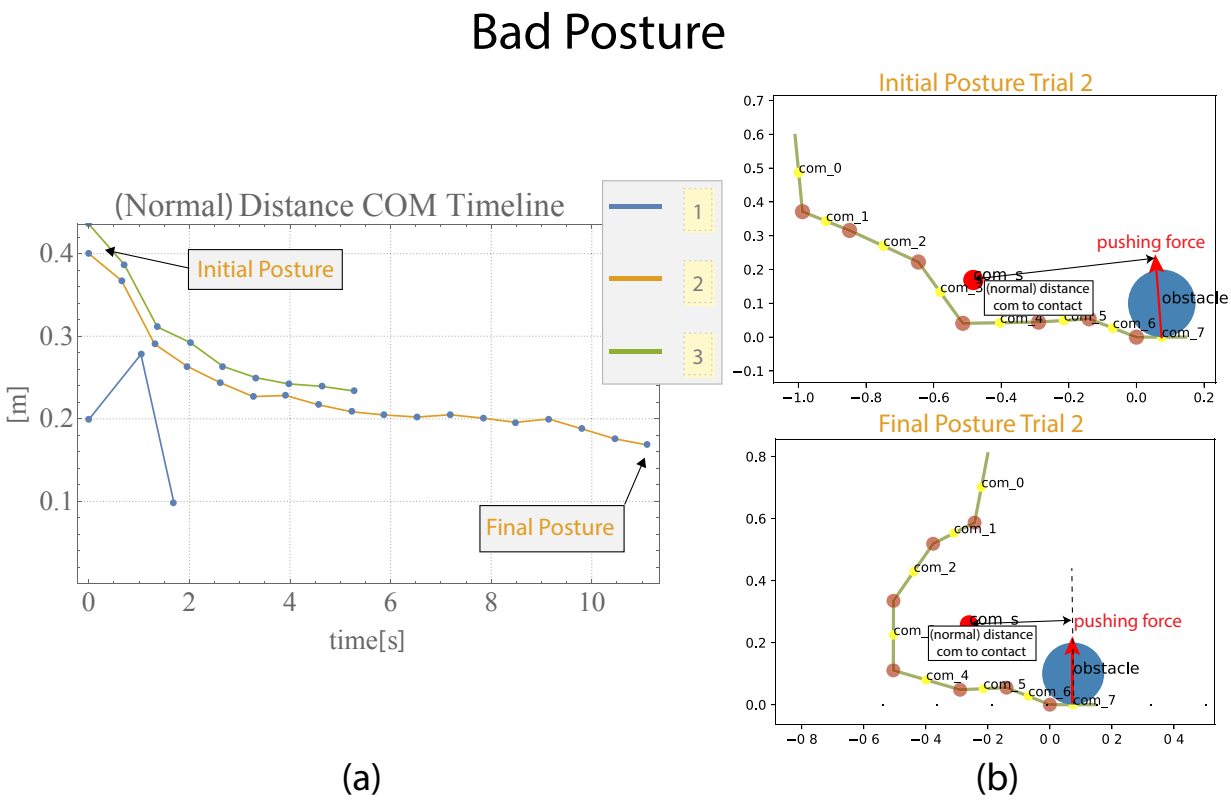


FIGURE 4.6: **Results experiment 2 - A snake robot with bad posture pushing against an object**
(a) The time history of the normal distance from the COM of the snake robot to the pushing line for three trials (b) The initial and final robot's posture for the second trial

In all experiments, it is interesting to see how the system evolves while solving an optimization problem. It is important to remark again, that the optimization problem tries to maximize the slippage ratio (2.84) which is a trade-off between the motion of the object and the robot itself. In other words, it would be possible to instead try to maximize the acceleration of the object directly, but this would result in a high slippage of the robot, making it more difficult to maintain control while performing a task.

The force sensor readings were synchronized with the robot. It is possible to obtain the magnitude of the force imparted onto the obstacle. Notice that, because the link of the robot is a simple planar object, it is always possible to ascertain that the direction of the pushing force is perpendicular to the snake robot. This is possible because, as mentioned in Section 4.1, the force sensor is mounted on a bearing, allowing free rotation. If there were friction between the link and obstacle, then it would be more difficult to know the direction of the force.

Fig. 4.8 shows the time history of the force sensor readings for all setups and trials. Fig. 4.8(a) corresponds to the first setup: Good posture while pushing. It can be seen that the force is almost constant throughout the trial. On the right, a histogram shows that, aside from trial 1, the statistical distribution of of the force is very similar. Forces tend to group around 3[N]. However, the robot eventually looses contact with the obstacle. Fig. 4.8(b) corresponds to the second setup: Bad posture. Trial 1 ended early because the robot lost contact. The other two

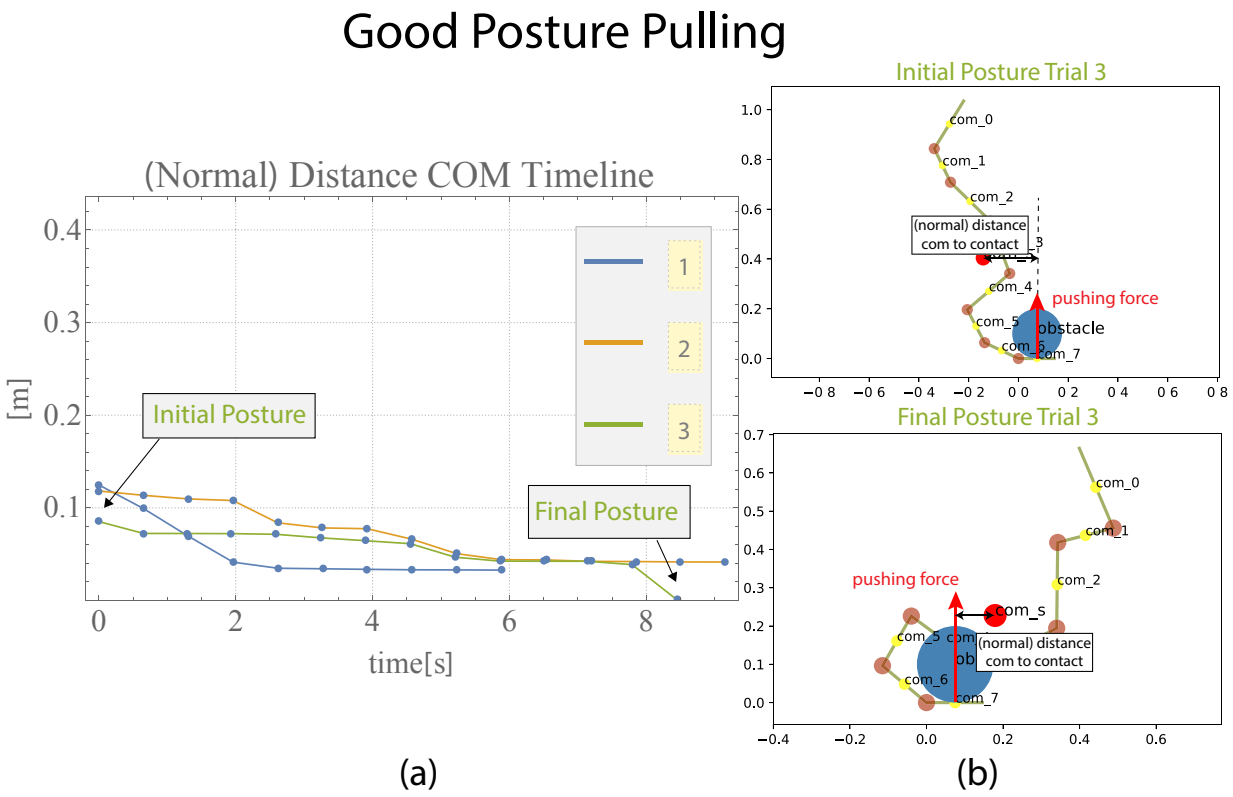


FIGURE 4.7: **Results experiment 3 - A snake robot with good posture pulling an object**
(a) The time history of the normal distance from the COM of the snake robot to the pushing line for three trials (b) The initial and final robot's posture for the third trial

trials showed very similar responses. The maximum force tended to group around 2[N]. Fig. 4.8(c) corresponds to the third setup: Good posture while pulling. All trials showed very similar responses. In all cases, the maximum force grouped a little lower than 3[N], very similar to setup 1.

It is interesting that all trials showed similar responses, even though the control law was calculated with an optimization problem, that may not give the same answer, even for similar initial conditions. As it can be seen, good postures allows the robot to impart a stronger force, almost 50% more than the bad posture. The best setup was pushing the obstacle with a good posture, however, the robot may loose contact eventually. On the other hand, pulling the obstacle seemed to give very consistent results. In fact, we consider that this can be extended in the future to enveloping grasps, as discussed in 2.4. Thanks to these trials, it is evident that the robot will naturally tend to some postures more than others. Pushing the object seems to lead to bad postures, while pulling leads to get the COM of the robot closer to the object. This can also be extended to climbing [56] or OAL [20]. As discussed in Chapter 2, Manipulation and Locomotion are mathematically related, and can be better understood under the framework of EAM.

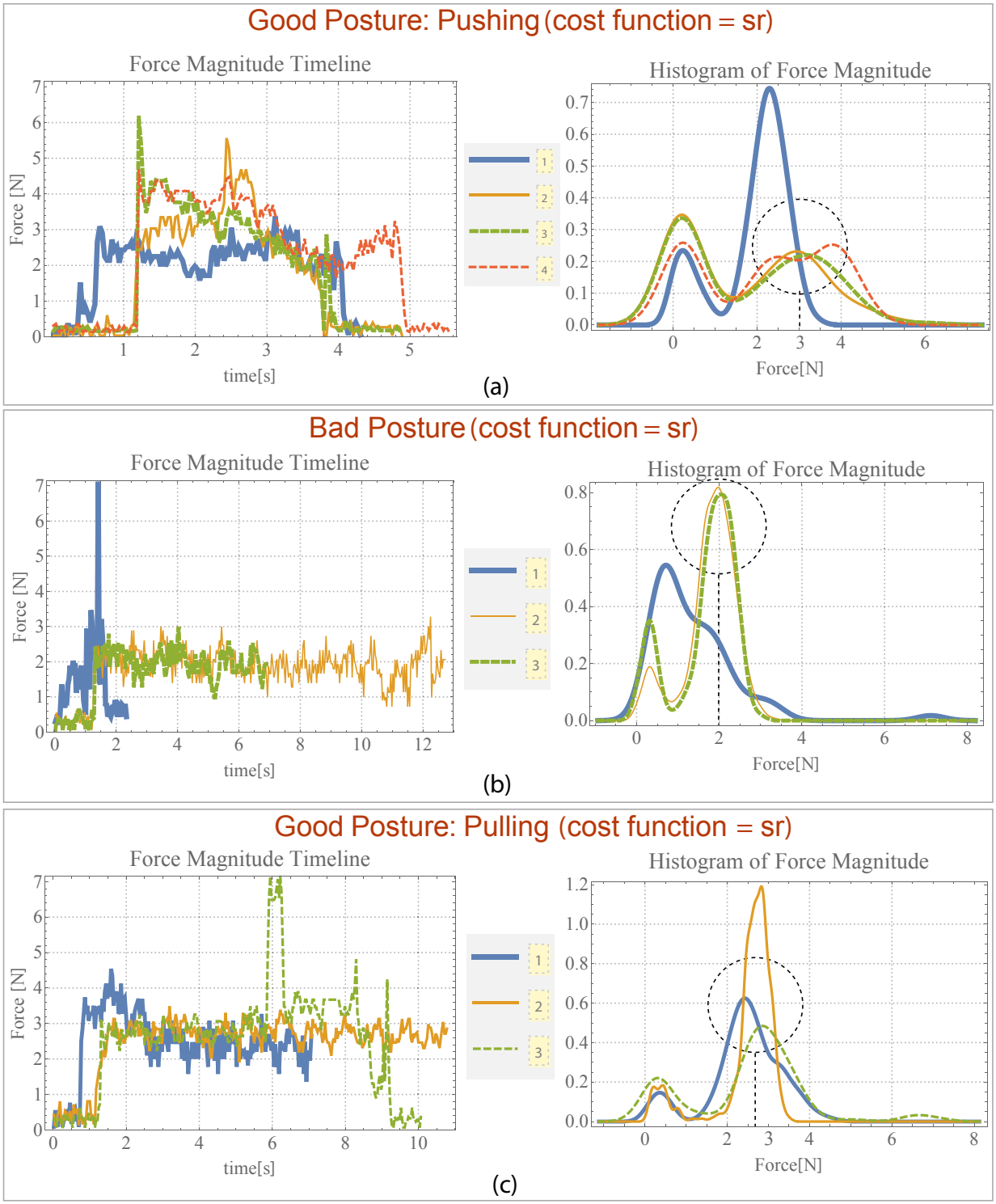


FIGURE 4.8: **Results experiments - Force applied to the obstacle**
(a) Good posture while pushing (b) Bad Posture (c) Good posture while pulling

Chapter 5

Discussion

In this thesis, a complete analysis of a snake robot interacting with an object has been presented. The analysis started with a kinematic analysis to consider feasibility of form-closure, but later extended to consider the inertial parameters of both robot and object. As it has been seen, the interaction of the snake robot with the ground, which is considered part of the environment, has a deep and meaningful impact on the acceleration of an object to be pushed. This interaction with the environment can be either helpful (it minimizes the slippage of the robot) or harmful (too much friction constrains the snake robot). The framework of **Environment-aided Manipulation (EAM)** presented in this thesis helps to elucidate this complex system.

Compared to literature discussed in Chapter 1, several parameters that had been ignored previously have been included in the analysis. This allows to provide conclusions and insights that would not be possible otherwise. For example, we made no assumptions that the effects of the passive wheels can be modeled as non-holonomic constraints, which is the most common model for studying snake robots. This assumption is equivalent to the Scenario 3 presented in Section 3.2. This is due to the fact that locomotion is usually studied, and a steady-state response can be obtained, since locomotion is a periodic input. However, for the topic of grasping with snake robots this assumption is not well suited, as it has been discussed in [24]. The scenarios presented in this thesis consider all possible cases of anisotropic friction: negligible, bounded, and ideal (unbounded). The results have the same tendency for all three scenarios.

The analysis presented can be applied to any planar snake robot, regardless of the number of joints. Additionally, as it has been shown in Chapter 3, the optimal postures of the snake robot to push an object are almost not influenced by factors like friction. On the other hand, the slippage of the snake robot is minimized when using passive wheels, as it has been shown in Section 3.4.

The analysis presented uses a general methodology to build the dynamical models of snake robot and object, in the framework of rigid-bodies. This is inspired in methods presented in [33, 36] among others. A great advantage of this is that the analysis is easily applicable to spatial motions. In other words, it can be expanded to consider snake robots in 3D motion, not necessarily confined to a plane.

Although the analysis presented has been rigorous, there are several questions that remain unanswered. Section 3.3 and Section 3.4 show that the set of postures optimal for pushing an object

and minimizing the slippage of the snake robot are not the same. Additionally, it can be seen in Fig. 3.5 that the input is not the same. Fig. 3.6 shows that posture plays an important role in maximizing the force imparted onto the object, and therefore in maximizing the resulting acceleration. However, there is no cleat set of postures that maximize the slippage ratio. We consider this to be the first step to formulate a strategy. A weighted combination can be considered to choose between maximizing the acceleration of the object, or minimizing the slippage of the robot. This is similar to how applications for robotic manipulators have benefited from the knowledge of their inertial properties.

In Section 3.3 snake robots with two, three, and four joints were considered. All case studies showed the same results. Postures where the COM is close to the line of action of pushing are optimal, regardless of the number of joints. We consider this to be an important result, since it can be extended to a continuous snake robot (i.e. $n_a \to \infty$), where rigorous modeling may not possible. In other words, this thesis shows that it the particular configuration of the robot (i.e., the set of joints' angles) is not the primary concern but rather the overall shape of the snake robot, in other words, its **posture**. It is the inertial properties of the posture that have a significant impact on the system.

Since the analysis is done using general differential geometry and algebra, the mappings and metrics presented can be applied to any robotic system with similar features. We consider this to be an extension of the concept of *force ellipsoid* or even manipulability metrics [38, 63, 69, 70, 67, 68] which sometimes ignore either mobile robots or the effect of inertial parameters.

A limitation of the models presented so far is that the results are limited to planar snake robots on a horizontal plane. Under other circumstances, the effect of gravity must be considered as an external force. This can be done easily by projecting the gravity onto the plane, and considering its normal and tangential components, as can be seen in Fig. 2.4. In the presence of gravity, it would be necessary to consider the effects on the constraint forces. For example, if the snake robot is pushing against the object on a slope, then gravity would be affecting the object, which in turn would be pushing against the robot.

Although the primary concern of this thesis has been to study a snake robot pushing against an object, the next logical step would be to extend the results to a more complicated manipulation, like dexterous (non-prehensile) manipulation and prehensile manipulation. Although sufficient conditions for grasps with form-closure have been presented, the next step would be to study force-closure. To do this, the set of forces that the snake robot can apply must be studied. This is a challenging topic, since the snake robot is under actuated. However, the study about **slippage ratio** would be helpful to extend this concept, since it is a metric that quantifies the motion of the system. The lower the value of the slippage ratio, the more the snake robot slips.

The framework of EAM is intended to study several systems under a unifying framework, as discussed in Chapter 2. More and more effects have to be considered, but not without taking into consideration the task to be fulfilled. Obviously, more complicated friction models could be considered. However, more complicated does not equate to useful. Snake robots have to be

used in more diverse tasks than locomotion, in order to understand what type of models and control strategies are necessary.

The main drawback of the EAM framework presented in this thesis, is that it relies on modeling all bodies on the system as rigid bodies. This ensures that a feasible and unique solution can be found. However, it limits the number of bodies in contact and may not be suitable for more realistic scenarios. If more contacts are considered, for example, between the snake robot and ground or one link contacting an obstacle at several points, the system may be hyper-static [40]. One way to avoid this is to consider compliance at the contacts. However, estimating this compliance is not trivial, and it may be difficult without knowing very precisely the properties of the snake robot, environment, and object to be manipulated. The framework of EAM discussed here can be used as a basic case to be compared to.

Chapter 6

Conclusions & Future Work

6.1 Conclusion

In this thesis a system composed of a planar snake robot and object has been presented. The objective is to study several parameters that may influence a manipulation task.

It is shown that the optimal configurations to push an object are the ones that have the COM of the robot along the action line. These configurations also minimize the slippage of the snake robot. However, although the configurations are the same, the inputs are not. A trade off must take place. Either the acceleration of the object is maximized, or the robot's slippage minimized. An analytical model is presented that shows some insights into this interaction. The simplified model can qualitatively explain why these configurations are optimal.

Additionally, it has been shown that additional friction forces between the snake robot and ground have very little impact on the acceleration of the object to be manipulated. However, these friction forces can help to reduce the slippage of the snake robot.

Finally, a control law based on an optimization problem is proposed, and experimental results are shown. Both mathematical analysis and experimental results show that postures where the COM is aligned with he action line minimize the slippage of the robot, while allowing to push the object better.

6.2 Future Work

As it has been shown, the interaction with the object can be characterized by the constraint forces imposed between the object and snake robot. If it is the objective to move the object, a set of joint torques (i.e., the input to the system) has to be selected in orderer to maximize the acceleration of the object. Howe ever, the model and analysis presented are not limited to manipulation. The framework of EAM can be extended to locomotion.

For example, if it is necessary to move around an object while not damaging it, then the constraint forces (and consequentially the acceleration of the object) have to be minimized. At the

same time, if locomotion is the objective, maximizing the motion of the robot is necessary. These objectives are dual to the objectives of manipulation.

Manipulation requires to maximize the acceleration of the object and minimize the slippage of the robot. This can be simplified as maximizing the slippage ratio proposed in this thesis.

On the other hand, locomotion may be formulated as minimizing the forces imparted onto the object while maximizing the motion of the robot. This could be simplified as minimizing the slippage ratio.

Cases where a combination is necessary can also be studied. For example, climbing [56] requires to grasp the environment very strongly and then using this interaction to propel the robot in a desired direction. This can also be considered similar to Obstacle-aided locomotion (OAL) [20, 21, 22].

All these cases are one specific case of EAM. The next logical direction for this study, is to then use the knowledge presented in this thesis to formulate better control strategies. The control strategy proposed in this paper is but one of many possible choices. In particular, the optimization problem (4.2) handles the *slippage ratio* as a cost function to be maximized. Another point of view taken in [24] and [22] is to propose the minimization of input torques, while formulating the control objectives as (soft) constraints. Although this seems theoretically correct, the evolution of the system must also be accounted. In [22] the evolution of the system was not studied so it is not possible to ascertain the performance under that particular control law.

Appendix A

Mathematical Background

In this section, we give a very brief introduction to the mathematical topics necessary to understand this paper. We recommend [33, 34] for a more detailed treatment. The foundations of the model used in this paper have been presented in [24, 25, 26]; readers are encouraged to read this reference for a more detailed treatment of snake robots in the framework of articulated-bodies.

A.1 Differential Geometry

Assume a body B is to be studied. Its twist $\boldsymbol{v} \in \Re^{6\times 1}$ (concatenation of linear and angular velocity) and a wrench $\boldsymbol{f} \in \Re^{6\times 1}$ (concatenation of linear force and torque) acting on the body can be expressed w.r.t to any reference frame, not necessarily attached to the COM of the body.

To express the quantities w.r.t. a specific frame, the following transformations can be used. Let's denote with a superscript the point where a twist or wrench are expressed (e.g., ${}^{A}\boldsymbol{v}$ denotes the twist $\boldsymbol{v}$ at frame A). The following transformations can be used for changing from reference frame $\{A\}$ to frame $\{B\}$ as

$$ {}^{B}\boldsymbol{v} = {}^{B}\boldsymbol{X}_{A}\,{}^{A}\boldsymbol{v}, \qquad\qquad {}^{B}\boldsymbol{f} = {}^{B}\boldsymbol{X}^{*}_{A}\,{}^{A}\boldsymbol{f}, $$

with the transformations for twists and wrenches defined as

$$ {}^{B}\boldsymbol{X}_{A} := \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{r}^{\times} & \boldsymbol{E} \end{bmatrix}, \qquad\qquad {}^{B}\boldsymbol{X}^{*}_{A} := \begin{bmatrix} \boldsymbol{E} & -\boldsymbol{E}\boldsymbol{r}^{\times} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix}, $$

where $\boldsymbol{E} \in \mathrm{SO}(3)$ denotes the rotation matrix required to change frame $\{A\}$ into frame $\{B\}$, $\boldsymbol{r} \in \Re^{3\times 1}$ is the translation vector from the origin of frame $\{A\}$ to the origin of frame $\{B\}$ ($\boldsymbol{r}$ is expressed w.r.t. frame $\{A\}$), and $\boldsymbol{r}^{\times} \in \Re^{3\times 3}$ denotes the cross-product operator (also called skew-operator [35, 36]) for a vector $\boldsymbol{r} = [r_x, r_y, r_z]$ with the property $(\boldsymbol{r}^{\times})^{T} = -\boldsymbol{r}^{\times}$. Another useful fact that will be required later is the norm of the vector $\boldsymbol{r}^{\times}\boldsymbol{a}$ defined as

$$ ||\boldsymbol{r}^{\times}\boldsymbol{a}|| = ||\boldsymbol{r} \times \boldsymbol{a}|| = ||\boldsymbol{r}||||\boldsymbol{a}||\sin(\theta)|, $$

for any vector $\boldsymbol{a}$, where the angle θ is the angle between $\boldsymbol{r}$ and $\boldsymbol{a}$.

A.2 Dynamic Modeling of Objects

A rigid-body is able to move in its operational space with dimensions $n_{op} \in \mathbb{N}$ and its equations of motion can be compactly written as

$$\boldsymbol{I}_{obj}\boldsymbol{a}_{obj} + \boldsymbol{p}_{obj} = \boldsymbol{f}_{obj}, \tag{A.1}$$

where $\boldsymbol{a}_{obj}$, $\boldsymbol{p}_{obj}$, and $\boldsymbol{f}_{obj} \in \Re^{n_{op}}$ denote the acceleration, velocity-produced terms, and total wrench acting on the body, respectively. The inertia tensor w.r.t the COM of the object can always be factorized as proportional to the mass as

$$\boldsymbol{I}_{obj} = m_{obj}\bar{\boldsymbol{I}}_{obj}, \qquad \boldsymbol{I}_{obj}^{-1} = \frac{1}{m_{obj}}\begin{bmatrix} \bar{\boldsymbol{I}}_{com}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}. \tag{A.2}$$

The inertia tensor $\bar{\boldsymbol{I}}_{obj}$ represents the inertia of a body with unitary mass (i.e., $m_{obj} = 1$) and is defined as

$$\bar{\boldsymbol{I}}_{obj} := \begin{bmatrix} \bar{\boldsymbol{I}}_{com} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}, \tag{A.3}$$

where $\bar{\boldsymbol{I}}_{com}$ denotes the rotational inertia of the object when $m_{obj} = 1$. This factorization is an important fact that will be exploited in later formulations. More details about the spatial inertia tensor can be found in [36].

A.3 Metric Tensors and Norms

It is important to notice that, a generic system that is not composed of point-masses, may not be orthogonal. This means that the common definition of norm of a twist $||\boldsymbol{v}||^2$ based on inner product (e.g., $||\boldsymbol{v}||^2 = \boldsymbol{v} \cdot \boldsymbol{v} = \boldsymbol{v}^T\boldsymbol{v}$) would give incorrect results.

Let's denote the metric tensor of the covariant basis as $\boldsymbol{I}$ and its inverse by $\boldsymbol{I}^{-1}$. The (squared) length of a twist and wrench is an invariant quantity and can be obtained using the scalar product $\{\circ\}$ while taking into account the metric tensor as

$$||\vec{\boldsymbol{v}}||^2 = \vec{\boldsymbol{v}} \circ \vec{\boldsymbol{v}} = \boldsymbol{v}^T\boldsymbol{I}\boldsymbol{v}, \tag{A.4}$$

$$||\vec{\boldsymbol{f}}||^2 = \vec{\boldsymbol{f}} \circ \vec{\boldsymbol{f}} = \boldsymbol{f}^T\boldsymbol{I}^{-1}\boldsymbol{f}. \tag{A.5}$$

A.4 Constraints

The system considered in this document is composed of several bodies that may or may not have contact between each other. The snake robot is modeled as an open kinematic chain. The constraints imposed by the joints are not modeled, since we use the kinematic model (i.e, the Geometric Jacobians 2.1) to build the dynamic model 2.43. This is consistent with a Lagrangian

Framework [37, 35, 38] and avoids the need to calculate the internal forces that keep the robot together.

On the other hand, when the snake robot contacts an object, a set of non-penetration constraints appear that keep the bodies from penetrating each other. These constraints, also called *kinematic constraints* or *body constraints* are introduced in Appendix A.4.1 and explained in more detail in Appendix 2.3. The forces that keep the bodies from penetrating each other have to be calculated [33, 37, 39, 40, 41] and added to the system.

A.4.1 Kinematic Constraints Between Rigid Bodies

Assume two systems B_1 and B_2 are contacting each other at several points, imposing $n_{c,np} \in \mathbb{N}$ non-penetration (also called *kinematic* or *body*) constraints

$$\boldsymbol{A}_b \boldsymbol{v} \geqslant \boldsymbol{0}, \tag{A.6}$$

$$\boldsymbol{A}_b := \boldsymbol{T}_b^T \begin{bmatrix} -{}^{bc}\boldsymbol{X}_1 & {}^{bc}\boldsymbol{X}_2 \end{bmatrix} \qquad \boldsymbol{v} := \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{bmatrix}$$

where $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ denote the twists of B_1 and B_2, respectively. The transformations ${}^{bc}\boldsymbol{X}_1$, ${}^{bc}\boldsymbol{X}_2$ change the twists of B_1 and B_2 from their local frames to the contact points (*bc* for body-constraint). The matrix $\boldsymbol{T}_b$ spans the directions where motion is constrained (e.g., for a frictionless contact point, it spans the normal direction at the contact). The constraints denote that rigid bodies cannot penetrate or pull each other, only push. The matrix $\boldsymbol{A}_b \in \Re^{n_{c,np} \times n}$ is usually called *constraint matrix* and it is a projection from the velocities of the system onto a subspace spanned by the constraints, referred to as *constrained subspace*.

A.4.2 Friction Constraints of the Passive Wheels

Friction is a complicated topic since several models exist to model it, but in practice it is unrealistic to know it accurately *a priori*. Furthermore, friction can be considered as kinetic friction or static friction depending if both surfaces are sliding w.r.t. each other or not, respectively. In the case of kinetic friction, it can be incorporated fairly easily into the model of the snake robot as an external force [8, 42]. However, static friction has to be modeled as a constraint force. This introduces several problems for snake robots since there are too many contacts between the belly of the snake robot and the ground, rendering the problem statically indeterminate.

It is common practice to introduce mechanical means for snake robots to have anisotropic friction at the contacts with the ground. In other words, the friction parallel to the links should be very low and in the perpendicular direction very high. In fact, this has been proven to be a very important property to ensure that snake robots have the ability to locomote [43, 42]. This is usually done by introducing passive wheels with their respective non-slippage constraints.

However, these constraints are usually presented in their unbounded form. In other words, it is assumed that the static friction forces can achieve arbitrarily high values and the total DOFs of the snake robot remain constrained [44, 45, 46]. From a mathematical point of view, this eliminates the need to analyze the problem dynamically, since the system is not supposed to gain momentum in the constrained directions.

Although this seems to give good enough results for locomotion or other periodic motions, it may not be the same for other tasks. It is interesting then, to consider the limit surfaces of the friction forces [37]. In this paper, we assume the snake robot has anisotropic friction with the ground and consider only friction in the direction perpendicular to the link.

The constraints due to static friction can be constructed similarly to the kinematic constraints presented on the previous section. Let's assume that the i-th link of the snake robot is contacting the floor with a passive wheel which imposes friction in the direction perpendicular to the wheel, while allowing free motion on the direction parallel to the link. It is assumed that the passive wheel is located at the COM of the link (c.f. Fig. 2.4(c)), and that $n_w < n_\ell$ links have passive wheels which impose $n_{c,f} = n_w$ constraints due to friction. The constraints can be written compactly as

$$\boldsymbol{A}_f \boldsymbol{v} = \boldsymbol{0}, \tag{A.7}$$

$$\boldsymbol{A}_f := \boldsymbol{T}_f^T \begin{bmatrix} -{}^{fc}\boldsymbol{X}_1 & \boldsymbol{0} \end{bmatrix} \qquad \boldsymbol{v} := \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{bmatrix}.$$

The transformation ${}^{fc}\boldsymbol{X}_1$ changes the twist of B_1 from its local frames to the point where the friction is located (*fc* for friction constraint).

In this paper we assume there are no friction forces acting on links interacting with the object. This can be achieved by lifting slightly the links or by other mechanical means (e.g., retractable wheels or casters). This idea is similar to the *shape controllable points* introduced in [47, 44] which allow the snake robot to avoid singularities (i.e., not loose rank), or more recently, to the idea of dividing the snake robot into a part for locomotion and another one for a task [19].

A.4.3 Summary of Constraints

A complete system can be constructed by stacking as many constraints as there are on the system. If there are $n_{c,np} \in \mathbb{N}$ non-penetration constraints and $n_{c,f} \in \mathbb{N}$ constraints due to friction, the total constraint matrix and constraint forces can be obtained as

$$\boldsymbol{A}\boldsymbol{v} \geqslant \boldsymbol{0} \qquad \boldsymbol{f}_c := \boldsymbol{A}^T \boldsymbol{\lambda},$$
$$\boldsymbol{A} := \begin{bmatrix} \boldsymbol{A}_b \\ \boldsymbol{A}_f \end{bmatrix} \qquad \boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}_b \\ \boldsymbol{\lambda}_f \end{bmatrix}$$

where $\boldsymbol{A} \in \Re^{n_c \times n}$ is the constraint matrix including all the constraints, $n_c = n_{c,np} + n_{c,f}$ is the total number of constraints, n is the number of DOF on the system, and $\boldsymbol{\lambda} \in \Re^{n_c \times 1}$ is the vector of

constraint forces. Considering the definition of $\boldsymbol{A}_f$ in (A.7), the constraint matrix is a triangular matrix, since the friction forces acting on the snake robot do not constrain directly the object. This leads to the expression

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_b \\ \boldsymbol{A}_f \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{b1} & \boldsymbol{A}_{b2} \\ \boldsymbol{A}_{f1} & \boldsymbol{0} \end{bmatrix}, \tag{A.8}$$

This fact will be exploited in section 2.7.2 to obtain a simplified expression for the acceleration showing clearly the effect that the extra friction forces have.

Additional constraints are necessary, depending on the nature of the constraint forces

$$\boldsymbol{\lambda}_b \geqslant \boldsymbol{0}, \qquad\qquad |\boldsymbol{\lambda}_f| \leqslant m\mu_s||\boldsymbol{g}_n||, \tag{A.9}$$

where m_ℓ, g, and μ_s denote the mass of a link, gravity acceleration, and coefficient of static friction, respectively. Inequalities are meant to be component-wise.

The formulation used in this document to model the system in the framework of multi-body rigid systems is mainly based in [33, 48, 49]. Friction has also been considered, mainly using the formulations presented in [41, 36], which also discuss how to resolve the resulting optimization problem. Additional documents are [50, 51] that also discuss this topic.

In practical applications, the previously shown formulation of the constraints is either formulated as a Linear Complementary Problem (LCP) or as a Quadratic Problem [36, 35]. If it is necessary to implement the constraints in a simulation (i.e., forward dynamics) it is necessary to add some stabilization terms [41, 36, 50] in order to avoid errors due to numerical drift.

Appendix B

Prototype Design

This section provides documentation for the snake robot **OPPAS**, an open-source and parametric snake robot. The Appendix E has information about where to get the most up-to-date information and files.

The name **OPPAS** stands for **OP**pen-source **Pa**rametric **S**nake-robot. **OPPAS** is a snake robot intended for research with robots that mimic the structure of a snake. The main reason I started this project is because I wanted a fast way to produce a snake robot that was easy to build and deploy, but without the need of drastic changes if a component of the robot changes.

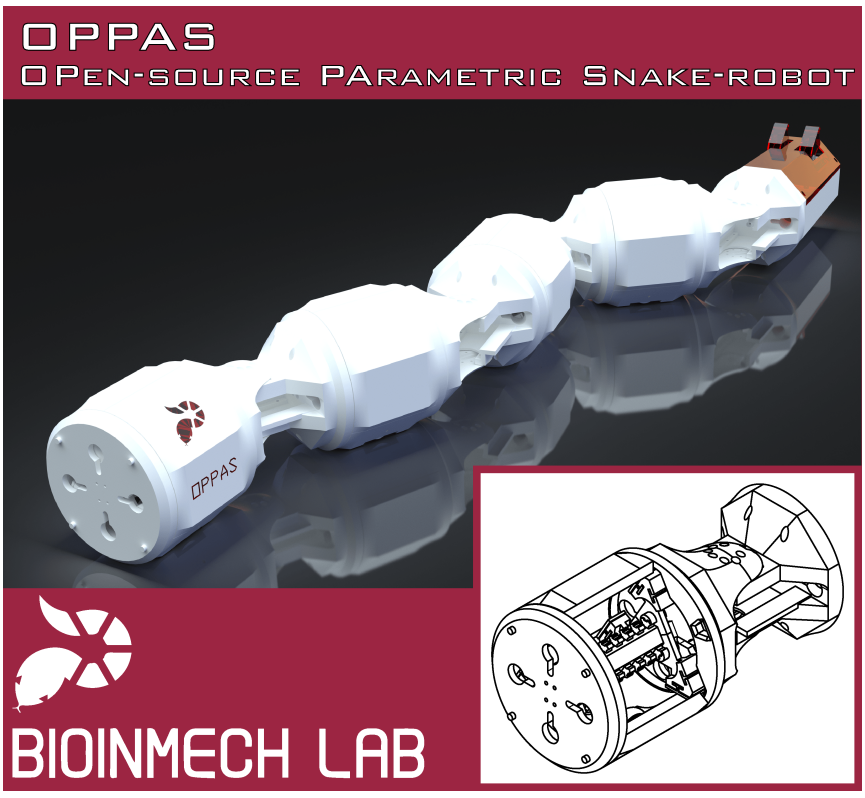


FIGURE B.1: **OPPAS** Snake robot prototype promotion

OPPAS has been presented in [31] at Fukushima, Japan.

OPPAS is a project being developed in Ritsumeikan University, by the Biomimetic Intelligent Mechatronics (BioInMech) Laboratory.

OPPAS is licensed under the Creative Commons Attribution 4.0 International. A summary of the license is provided below:

Listing B.1: CC BY 4.0

```
Share - copy and redistribute the material in any medium or format
Adapt - remix, transform, and build upon the material
for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license
   terms.

Under the following terms:

Attribution - You must give appropriate credit, provide a link to the
   license, and indicate if changes were made. You may do so in any
   reasonable manner, but not in any way that suggests the licensor endorses
    you or your use.

No additional restrictions - You may not apply legal terms or technological
   measures that legally restrict others from doing anything the license
   permits.
```

In essence, a snake robot is very simple; it is a series of links connected by joints. However, the shape and size of the links and joints will change depending on the components you use. For example, a big battery may require a long but slender link. But, depending on the microcontroller you use, you may need a short but thick link. Although putting together a bunch of u-brackets and other rigid parts is easy, this leaves the electronics and other components exposed, while at the same time properly fixing the components to the robot may be difficult. By making the design parametric, different snake robots can be produced, according to your requirements.

Another property I was interested in was modularity. By modularity I mean that a module (link + joint) of the snake robot is composed of different elements that can be interchanged without needing to redesign other parts. For example, if your snake robot requires passive wheels (e.g., for achieving anisotropic friction) just design the 'belly' of the robot without altering the rest. Or, you may change the wheel's model and redesign the necessary part. You are using a bigger battery that doesn't fit the current robot? Just make a longer link without affecting the joints.

However, there are some sacrifices to be made to keep the design as general as possible. Since the design is parametric (and the number of parameters is limited) the design is not optimal. In other words, there may be a lot of *wasted* space inside the robot.

OPPAS is an attempt to make snake robots easier to build and to test your ideas. It is **open-source**; the robot was designed using Autodesk Fusion 360. The files can be downloaded and the whole project can be recreated in your computer. I consider this to be more useful than sharing STL files that have to be downloaded and modified to fit your needs. As mentioned before,

OPPAS is **parametric**. Read the documentation for information about the available parameters. Finally, I wanted a robot that could be afforded by anyone. The current version of OPPAS has been printed in ABS using a Zortrax M200. Although the Zortrax may not be affordable by everyone, I have printed also some parts in my own Overlord Pro, and although the tolerances are more difficult to overcome, it still works. Each module costs less than US$ 20 (using eSun ABS).

There are several weak points in the current design of OPPAS. In particular, I would like to improve the following points:

1. **Waterproof**: Make the design waterproof
2. **Support for more servos**: Currently, OPPAS Mk. 1.0 is designed with the ROBOTIS MX-64 servos in mind. I would like to support more servo's models in the future, but parameterizing them is rather difficult (e.g., they have different shapes, the screw holes positions are in different arrangements)
3. **Smaller**: Due to the MX-64 servo's size, OPPAS Mk 1.0 is *big* in comparison to other snake robots. If possible, I would like to make it a little smaller next time

B.1 v1.0 - Locomotion Part

The main idea of OPPAS is to provide a platform for developers interested in snake robots or similar. A module of OPPAS consists of a link and a joint. A link and joint are further divided into the following parts (c.f. Fig. B.2):

The main idea of the design of parts was to separate the parts according to their role in the robot:

1. Skeleton: The spine, ribs and caps provide the main (kinematic) structure of the robot. It provides the main shape and can also be used as a base for the components.
2. Flesh: It provides the interface between the robot and the exterior. At the same time it protects the components (i.e., do not allow the boards and cables to be completely exposed).
3. Joint: The servos or other actuators are mounted at the joint, providing motion.

The link's main objective is to give structure to the robot and it has a big influence on the total length of the robot. In addition it will house the components (e.g., batteries, sensors, microcontroller boards). Depending on the robot you need, you may skip the link altogether and just put together a bunch of joints.

The link is divided into two sets of parts: the *skeleton* and the *flesh*. The main role of the skeleton is to provide the structure of the robot. The main parts of the skeleton are:

The design of the spine provides a flexible way to attach extra components. The idea is that you can design a specific base for the components you use, and then attach it to the spine. The spine has a set of rails and the geometry of these can be controlled parametrically. The final length of

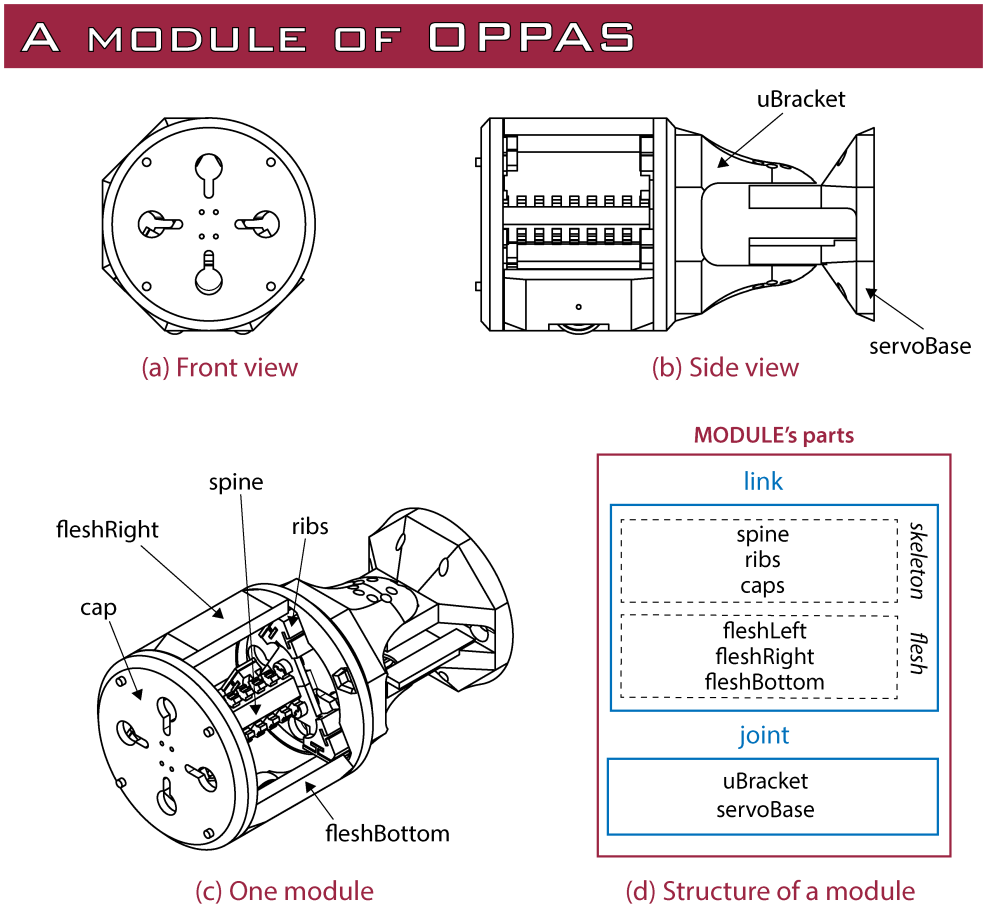


FIGURE B.2: **OPPAS Overview**
Main parts composing the snake robot

TABLE B.1: Parts of OPPAS

spine	Provides the main support for the link
ribs	Additional structure to attach the flesh components to the spine
caps	It helps to put all together and enclose the internal components. It also provides an interface with the joint's parts

the spine is controlled by the number of segments (rail's teeth + spacing) and the geometry of these.

The main parameters of the spine that can be changed are summarized as follows:

- `spineInnerDiameter`: The spine has an hexagonal cross-section, circumscribed on a circle with a diameter called `spineInnerDiameter`. A set of four M2.5 screw holes are automatically generated
- `rail*`: The set of parameters related to the rail design (`railNeck`, `railWidth`, `railInnerWidth`, `railHeight`) controls the geometry of the cross-section of the rail. The parameters `railSpacing` and `railThickness` control their geometry along the longitudinal axis of the spine. One 'segment' is composed of one rail's segment (I sometimes call them teeth) and a spacing.
- `noSegments`: Number of segments (rail's tooth + its spacing) composing the spine

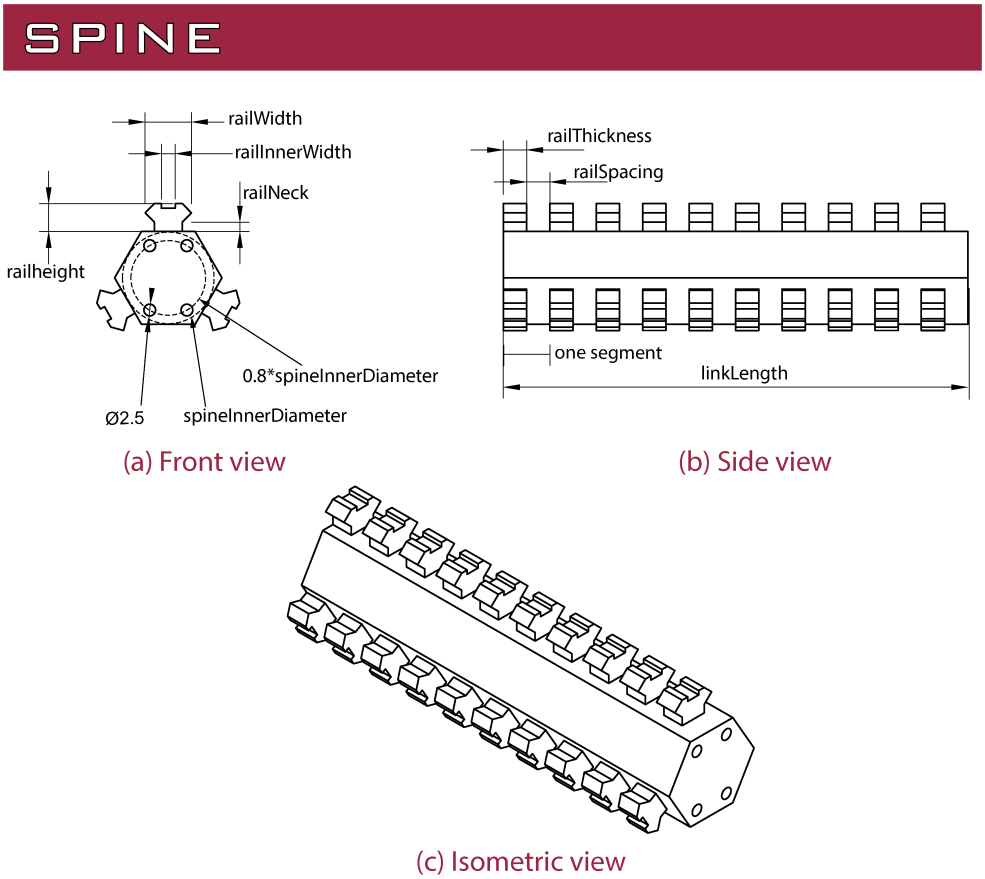


FIGURE B.3: **Spine of OPPAS**

- `linkLength`: The final length of the link is controlled by how many segments (a tooth and its spacing) you need, and its size controlled by `railSpacing` and `railThickness`. If you need a constant rail just set `railSpacing`= 0, `railThickness`=desired length, and `noSegments`=1. This allows to generate a link always consistent with a discrete number of teeth

The main role of the ribs is to attach the flesh components to the spine. A secondary objective I had into mind is to also provide a structure to attach extra components (e.g., sensors) if required.

Once the three components of the flesh slide into the ribs, their movement is constrained without the need of extra parts, like bolts or screws. The model of OPPAS that can be currently downloaded does not provide additional holes to attach the vertebras to the spine. It relies completely on friction and the form-closure achieved when the **flesh* parts are attached. This is to keep the design consistent and avoid problems, since a screw hole of a given size may not be compatible with the user's chosen `railThickness`.

The main parameters of the ribs that can be changed are summarized as follows:

- `midCircle`: This parameter is seen along several parts of OPPAS. For the ribs, it works along textttouterCircle to generate the structure of the ribs

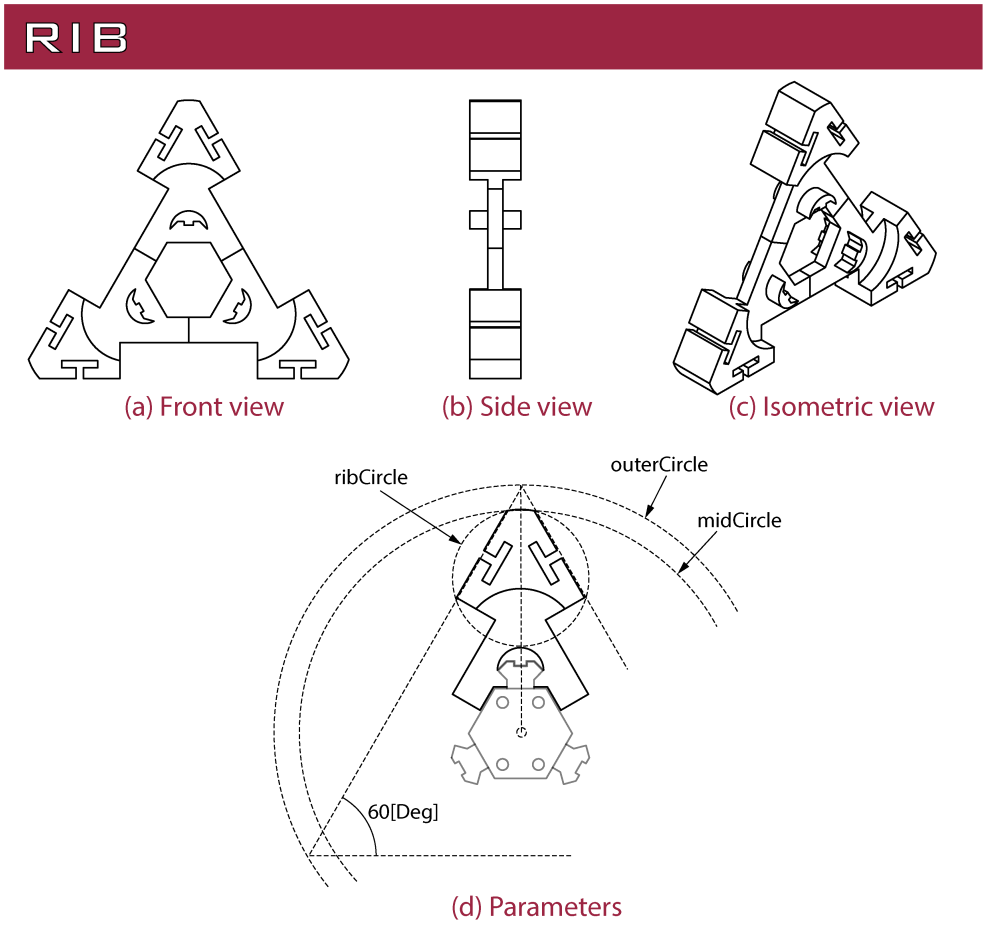


FIGURE B.4: **Ribs of OPPAS**

- `outerCircle`: This parameter controls the size of the ribs. An equilateral triangle is inscribed into a circle with diameter `outerCircle` and used to control the rib's size. The parameters `midCircle` and `outerCircle` should always follow the constraint: `|outerCircle| > |midCircle|`
- `ribCircle`: Final parameter to control the size of the rib. This parameter is not used in any other part of the design

The caps enclose the link and provide an interface with the joints. In addition, the interior of the caps accommodates to the ribs for extra support. Four slots are provided to pass cables, which location and size can be controlled by the user. Additional pegs provide extra strength to the design to avoid rotations along the longitudinal axis.

The main parameters of the caps that can be changed are summarized as follows:

- `capDiameter`: The outer diameter of the caps. It should be at least as big as `outerCircle`, but I recommend it a little bigger. In current implementation of OPPAS, the value is set as `capDiameter = 1.1*outerCircle`
- `outerCircle`: Outer circle will be the diameter of the cap contacting the joint's parts

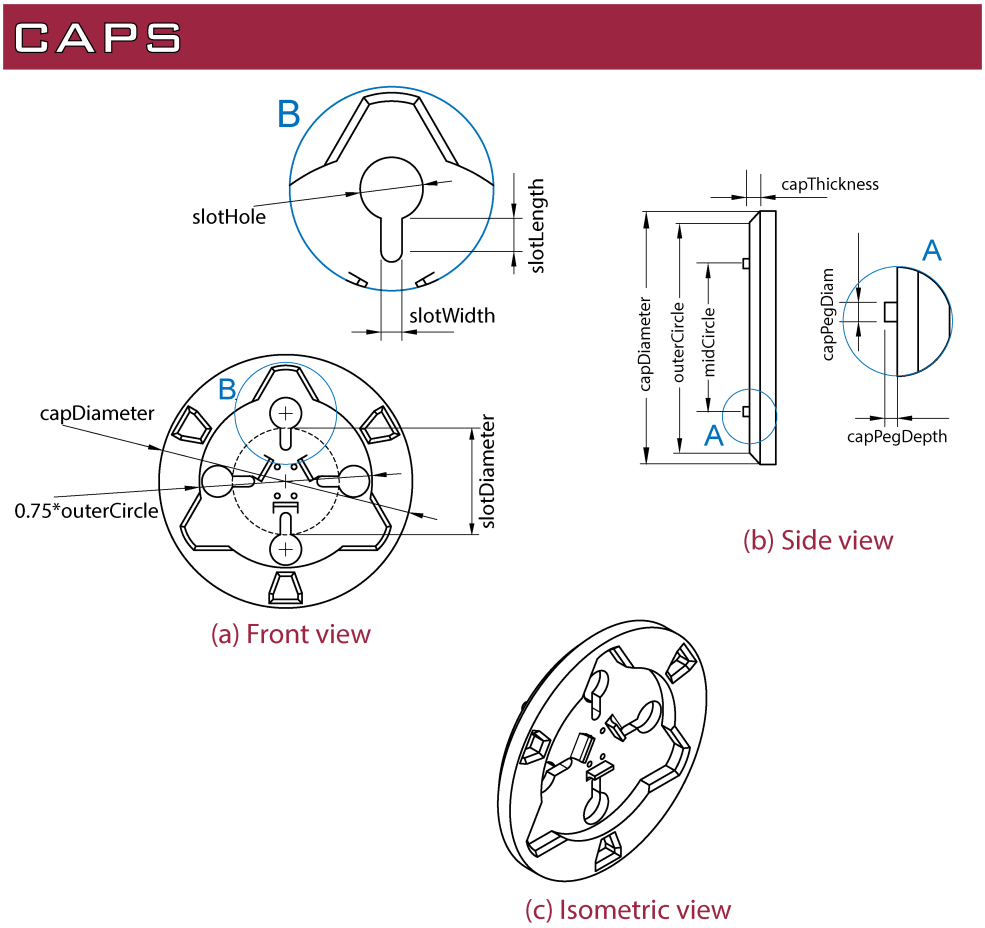


FIGURE B.5: **Caps of OPPAS**

- `capThickness`: Thickness of the cap (not considering the internal cavity where the ribs fit)
- `slotDiameter`: Control the location of the slots. In the current design of OPPAS, the parameter is set as `slotDiameter = 0.5*midCircle` to keep the design consistent. If required, change this to a fixed value
- `slotThikness`: Thickness of the slots
- `slotLength`: Length of the slots
- `slotHole`: Diameter of the slot's hole. Use this to pass cables with connectors. In general, I advice `slotHole` to be at least 10[mm]
- `midCircle`: A set of four pegs are generated automatically at a circle with this diameter. If needed, change this to a fixed value
- `capPegDiam`: Diameter of the pegs
- `capPegDepth`: Length of the pegs. I use 'depth' since a matching hole is generated into other parts

The *flesh* components are the main interface between the robot and the exterior. These parts can be designed independently from the rest of the design, but I provide a set of parts as an example. The *flesh* parts slide into the ribs and provide extra constraints. Once all parts of the flesh are inserted, the ribs should not be able to come apart from the spine.

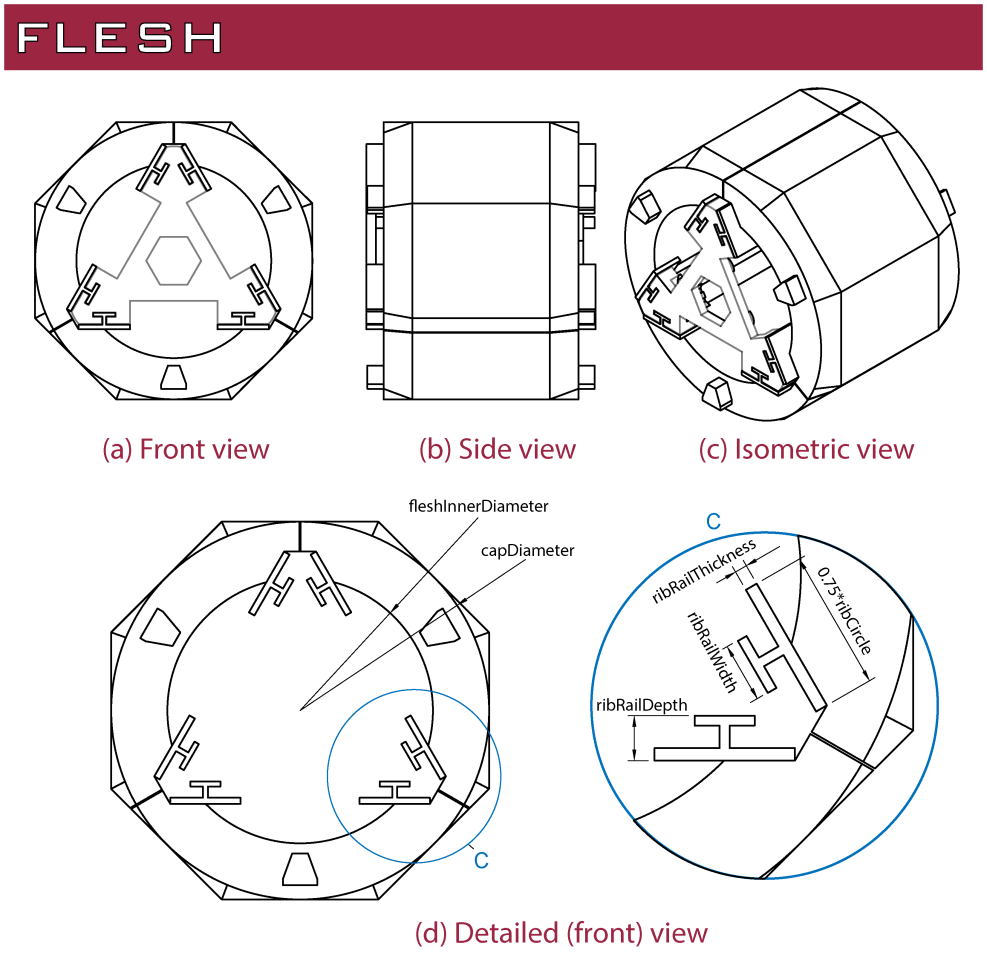


FIGURE B.6: **Flesh of OPPAS**

The main parameters of the flesh that can be changed are summarized as follows:

- `fleshInnerDiameter`: The inner diameter of the flesh parts is controlled by this parameter. The strength and available space inside the robot are affected. The outer size is the same as 'capDiameter'. You can change these values according to your application.
- `ribRail*`: The set of parameters (`ribRailThickness`, `ribRailWidth`, and `ribRailDepth`) control the geometry of the rails that slide into the ribs. There is a built-in tolerance for the rails, so that they fit correctly into the ribs even if there is shrinking of the material.

The *flesh* parts in **OPPAS** should not be considered as a definite design. The purpose of textbfOPPAS is that they can be redesigned to accommodate to your needs, without needing to change (drastically) the rest of the parts. Several designs of the *flesh* could be designed and used in the same base robot.

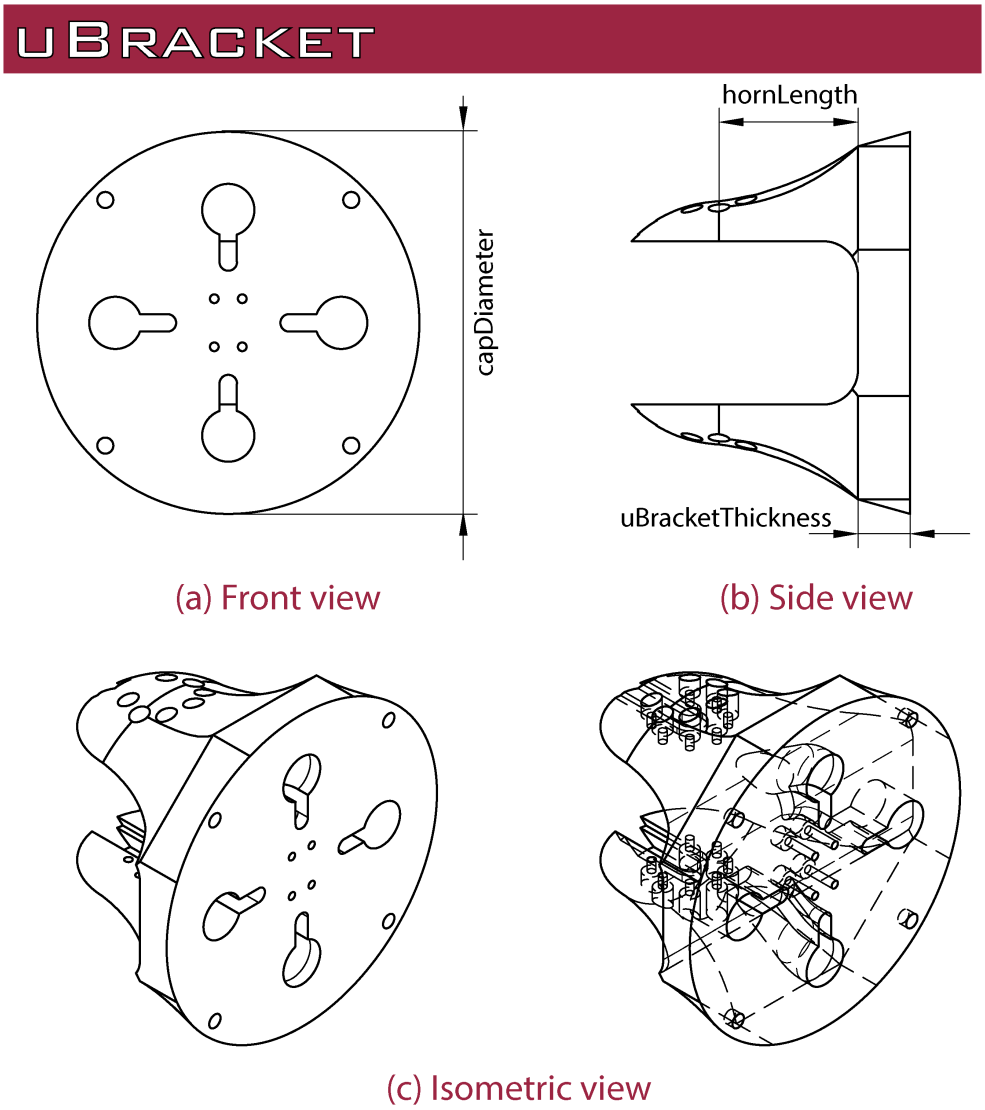


FIGURE B.7: **The uBracket**

The joint of is composed of two parts: *uBracket* and *servoBase*, as seen in Fig. B.7 and Fig. B.8, respectively. These are the parts that would require the most effort from the user if another servo is used.

The *uBracket* connects the servo's horn to the link, providing motion. There are four slots, designed so that the link can be connected with a 90[deg] angle to provide flexibility in the construction of your robot. If the axes of all servos are parallel, the result is what is known as 'planar snake robot'. What is known as 'serpentine' or 'undulatory' locomotion can be achieved. With the motors connected with an offset of 90[deg], 3D motions can be achieved.

The outer diameter of the uBracket is controlled by `capDiameter` and the slots are designed to match the ones in the module caps. The main parameters of the uBracket that can be changed are summarized as follows:

- `uBracketThickness`: Thickness of the uBracket.

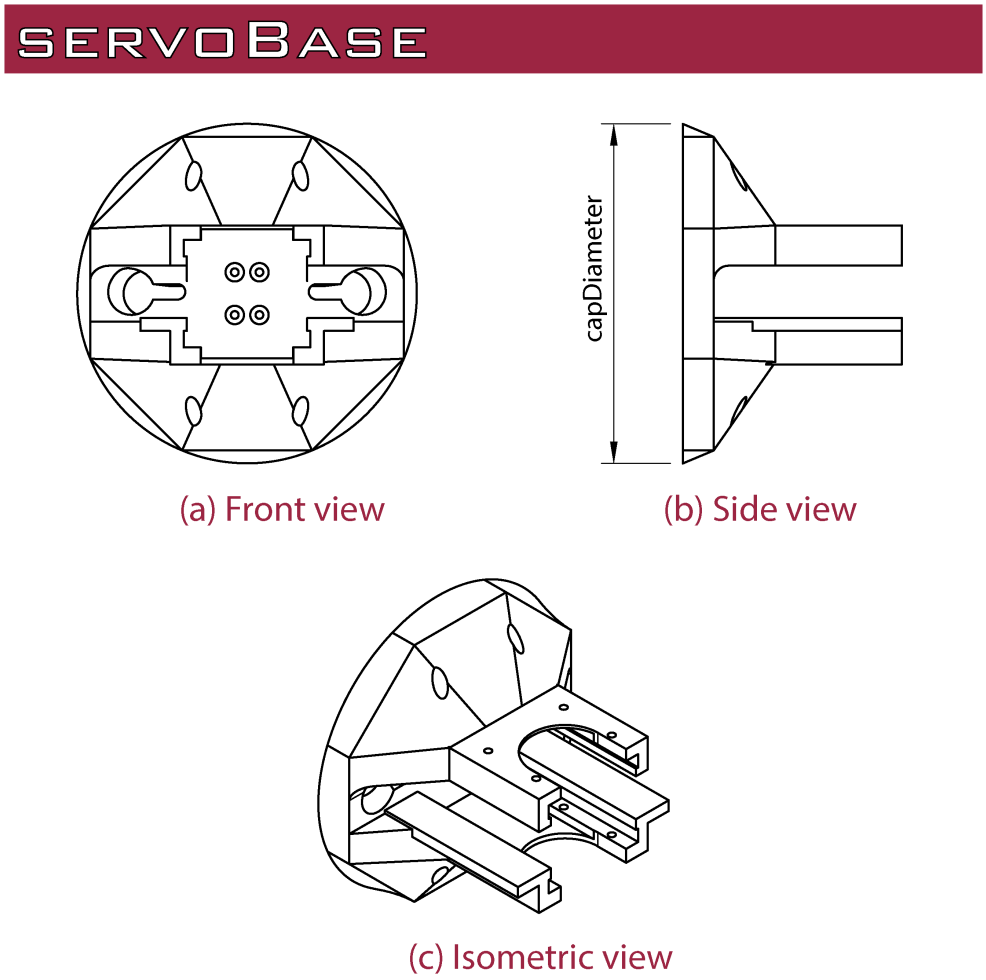


FIGURE B.8: **The servoBase**

- `hornLength`: Distance from the uBracket base to the axis of the servomotor. The longer `hornLength` is, the bigger the range of motion of the joint. However, this means a weaker uBracket.

The servoBase of the current design of OPPAS is designed for the MX-64AR servos of ROBOTIS. Therefore, no extra parameters are provided to the user.

The final module obtained can be seen in Fig. B.9. The values for the parameters are summarized in Fig. B.10.

B.2 v2.0 - Manipulation Part

TODO

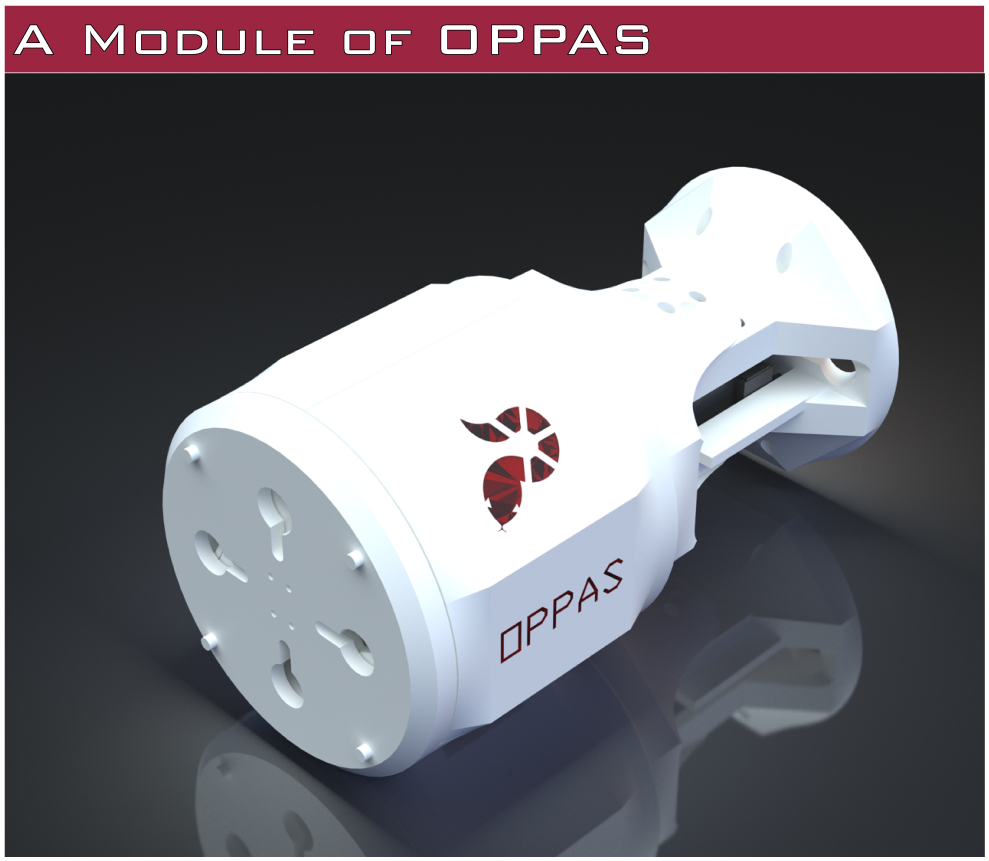
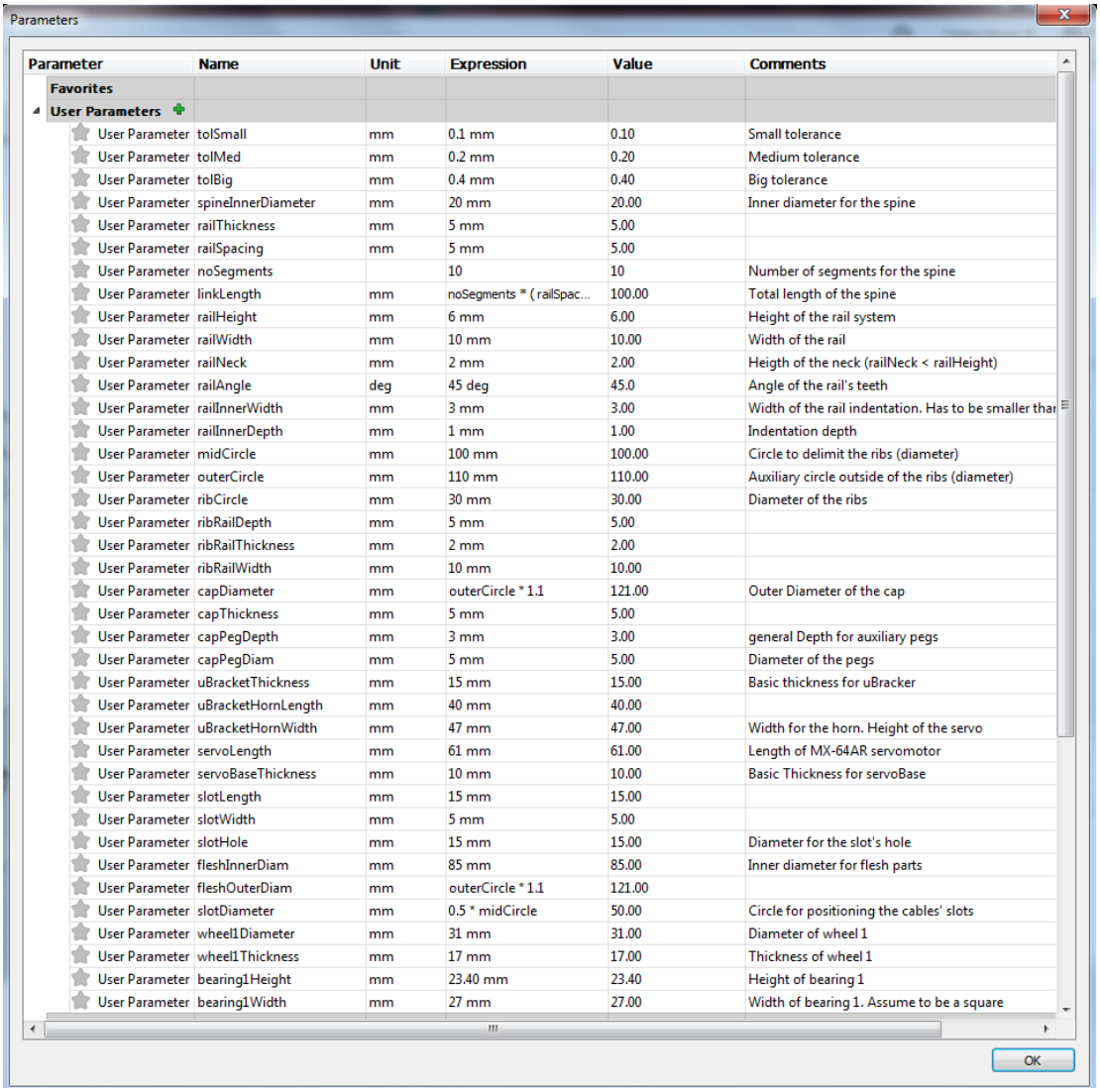


FIGURE B.9: **A module of OPPAS assembled**
An assembled module with the Biomimetic Intelligent Mechatronics Laboratory Logo ©



Parameters

Parameter	Name	Unit	Expression	Value	Comments
Favorites					
User Parameters					
User Parameter	tolSmall	mm	0.1 mm	0.10	Small tolerance
User Parameter	tolMed	mm	0.2 mm	0.20	Medium tolerance
User Parameter	tolBig	mm	0.4 mm	0.40	Big tolerance
User Parameter	spineInnerDiameter	mm	20 mm	20.00	Inner diameter for the spine
User Parameter	railThickness	mm	5 mm	5.00	
User Parameter	railSpacing	mm	5 mm	5.00	
User Parameter	noSegments		10	10	Number of segments for the spine
User Parameter	linkLength	mm	noSegments * (railSpac...	100.00	Total length of the spine
User Parameter	railHeight	mm	6 mm	6.00	Height of the rail system
User Parameter	railWidth	mm	10 mm	10.00	Width of the rail
User Parameter	railNeck	mm	2 mm	2.00	Heigth of the neck (railNeck < railHeight)
User Parameter	railAngle	deg	45 deg	45.0	Angle of the rail's teeth
User Parameter	railInnerWidth	mm	3 mm	3.00	Width of the rail indentation. Has to be smaller thar
User Parameter	railInnerDepth	mm	1 mm	1.00	Indentation depth
User Parameter	midCircle	mm	100 mm	100.00	Circle to delimit the ribs (diameter)
User Parameter	outerCircle	mm	110 mm	110.00	Auxiliary circle outside of the ribs (diameter)
User Parameter	ribCircle	mm	30 mm	30.00	Diameter of the ribs
User Parameter	ribRailDepth	mm	5 mm	5.00	
User Parameter	ribRailThickness	mm	2 mm	2.00	
User Parameter	ribRailWidth	mm	10 mm	10.00	
User Parameter	capDiameter	mm	outerCircle * 1.1	121.00	Outer Diameter of the cap
User Parameter	capThickness	mm	5 mm	5.00	
User Parameter	capPegDepth	mm	3 mm	3.00	general Depth for auxiliary pegs
User Parameter	capPegDiam	mm	5 mm	5.00	Diameter of the pegs
User Parameter	uBracketThickness	mm	15 mm	15.00	Basic thickness for uBracker
User Parameter	uBracketHornLength	mm	40 mm	40.00	
User Parameter	uBracketHornWidth	mm	47 mm	47.00	Width for the horn. Height of the servo
User Parameter	servoLength	mm	61 mm	61.00	Length of MX-64AR servomotor
User Parameter	servoBaseThickness	mm	10 mm	10.00	Basic Thickness for servoBase
User Parameter	slotLength	mm	15 mm	15.00	
User Parameter	slotWidth	mm	5 mm	5.00	
User Parameter	slotHole	mm	15 mm	15.00	Diameter for the slot's hole
User Parameter	fleshInnerDiam	mm	85 mm	85.00	Inner diameter for flesh parts
User Parameter	fleshOuterDiam	mm	outerCircle * 1.1	121.00	
User Parameter	slotDiameter	mm	0.5 * midCircle	50.00	Circle for positioning the cables' slots
User Parameter	wheel1Diameter	mm	31 mm	31.00	Diameter of wheel 1
User Parameter	wheel1Thickness	mm	17 mm	17.00	Thickness of wheel 1
User Parameter	bearing1Height	mm	23.40 mm	23.40	Height of bearing 1
User Parameter	bearing1Width	mm	27 mm	27.00	Width of bearing 1. Assume to be a square

OK

FIGURE B.10: **Parameters of first generation OPPAS**
Set of parameters used in the Autodesk ©Fusion 360 Software

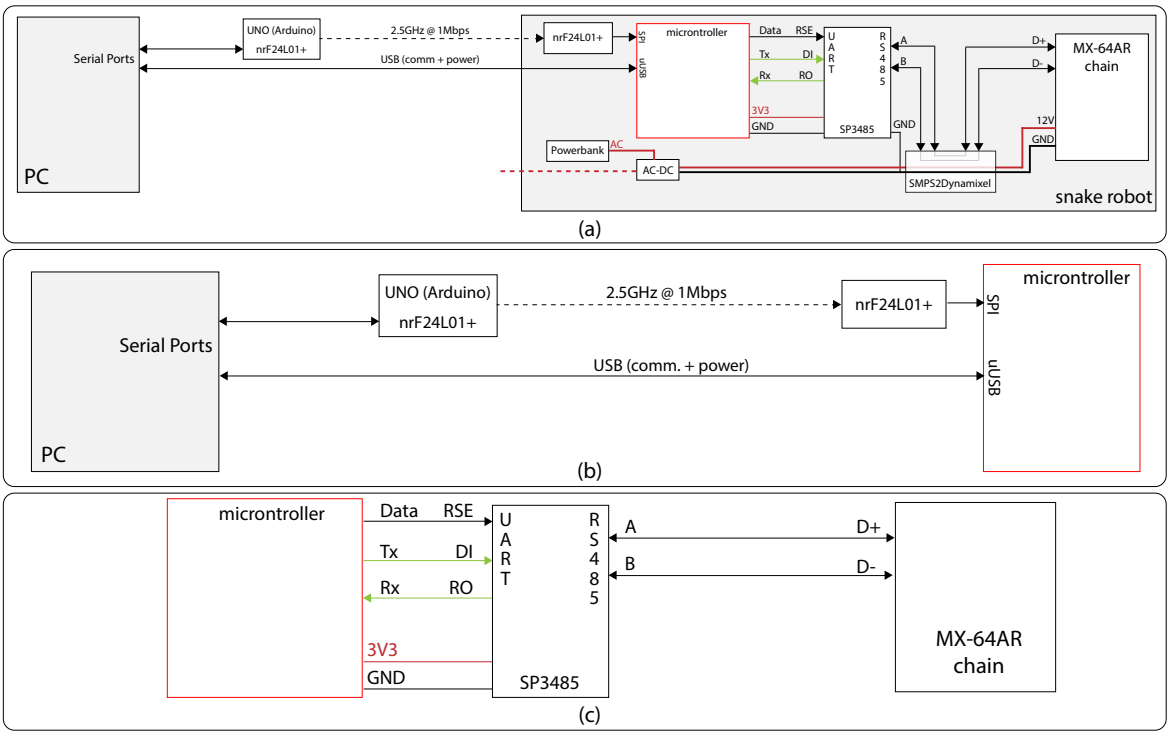


FIGURE C.1: **Complete System**
(a) Overview of the complete system. The snake robot can communicate with the PC either through USB or wirelessly using a nRF24L01+ transceiver (2.4[Ghz]) (b) Communication using either USB or nrF24L01+ (c) Communication with servomotors

Appendix C

Electronics

This section details the overall framework of the electronics of the prototype. The complete design including all signals can be seen in Fig. C.1.

Fig. C.1(b) focuses on the communication part of the prototype.

Fig. C.1(c) focuses on the communication between microcontroller and servomotors. Additional information about the software implementation can be found in Appendix D.1.

C.1 Shields - Communication with servos

The servomotors used in this prototype are Dynamixel model MX-64AR (Robotis `http://en.robotis.com/`). The servomotors are connected to the main microcontroller board Duo (Redbear `https://redbear.cc/duo`) through a RS-485 BUS. A transceiver is necessary to change the microcontroller TTL-level signals into the RS-485 levels.

A ST4485EB (STMicroelectronics `http://www.st.com/en/interfaces-and-transceivers/st4485eb.html`) was selected as the transceiver, and it is connected to the microcontroller as seen in Fig. C.1(c). It is worth noticing that the Duo is a 3.3[V] level micro controller board, since it uses an STM32F205 ARM 32-bit Cortex-M3 @120 MHz microcontroller (STMicroelectronics). If using a 5[V] microcontroller, a 5[V] transceiver is necessary. The SN65HVD1782DR (Texas Instruments `https://store.ti.com/SN65HVD1782DR.aspx`) has been tested using the library **DuoDMXL** described in D.1 The schematic design can be seen in Fig. C.2. Fig. C.3 shows the board design and a manufactured sample.

The EAGLE schematics and additional up-to-date information can be accessed through the public repositories listed in Appendix E.

C.2 softPots ADC conversion and conditioning

For measuring the contact location between the snake robot and an external object, a special type of potentiometers, called SoftPots (Spectra Symbol `http://www.spectrasymbol.com/`) were selected. The SoftPots resistance changes linearly w.r.t. the position where they are pressed, as it can be seen in Fig. C.4(a). Then the position where the SoftPot is being pressed can be calculated by measuring the resulting voltage in a simple voltage divider configuration. However, the SoftPots by themselves have two problems:

- Short-circuit: If the SoftPot is pressed at more than one places at the same time, a short-circuit occurs and the SoftPot may burn.
- Floating signal: If nothing is touching the SoftPot, then the signal will float unpredictably. In other words, it cannot be determined if there is contact or not.

These two problems can be solved easily, as it can be seen in Fig. C.4(a). The short-circuit can be avoided by limiting the current with in-series resistors R1 and R2. R2 is chosen to be a trimmer potentiometer (trimpot) in order to use it for more precise calibration. The floating signal can be avoided using a weak pull-up resistor R3. This will ensure that, when there is no contact, the signal will saturate to the input voltage VIN.

The use of R1, R2, and R3 scales the SoftPot signal and the signal is no longer linear. However, the new output signal can be calculated using the following formulas:

$$\mathrm{V}_{out} = \frac{\mathrm{RB}}{\mathrm{RA} + \mathrm{RB}}\mathrm{VIN}, \tag{C.1}$$

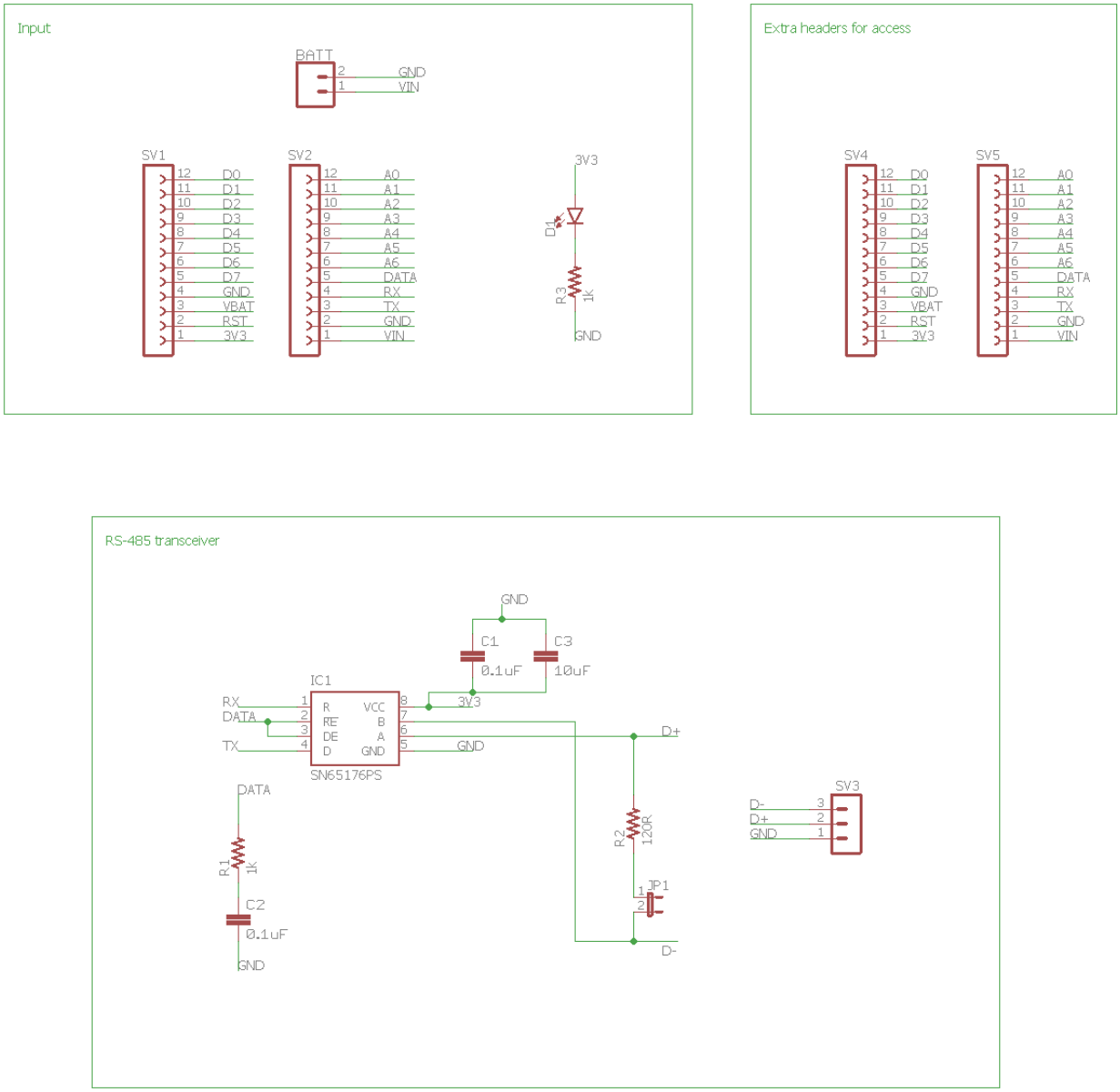


FIGURE C.2: **RS-485 Transceiver**
EAGLE Schematic of RR-485 transceiver circuit used to interface with servomotors MX-64AR

where

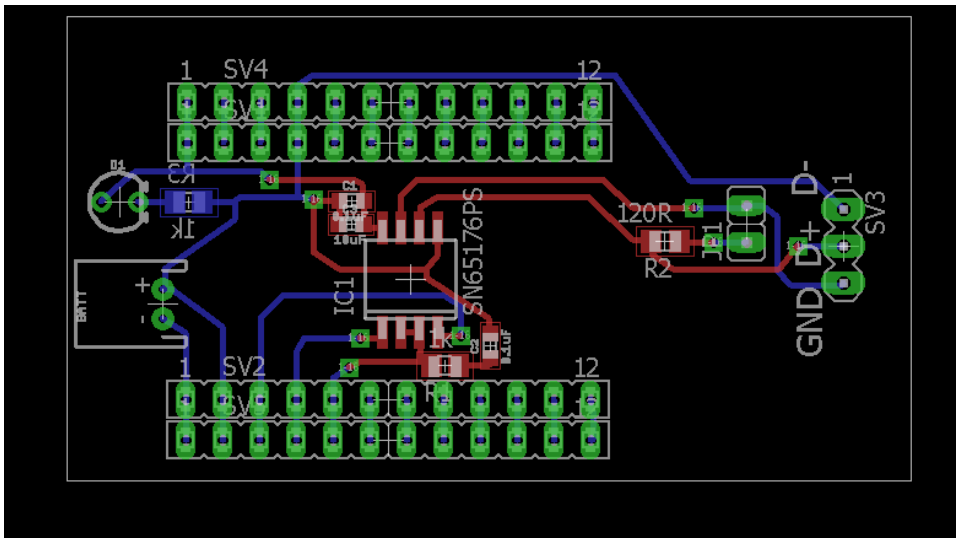
$$\mathrm{RA} := \left(\frac{1}{\mathrm{R1}+\alpha\mathrm{SP}} + \frac{1}{\mathrm{R3}}\right)^{-1} = \frac{\mathrm{R1R3}+\alpha\mathrm{R3SP}}{\mathrm{R1}+\mathrm{R3}+\alpha\mathrm{SP}}, \tag{C.2}$$

$$\mathrm{RB} := \beta\mathrm{SP} + \mathrm{R2}. \tag{C.3}$$

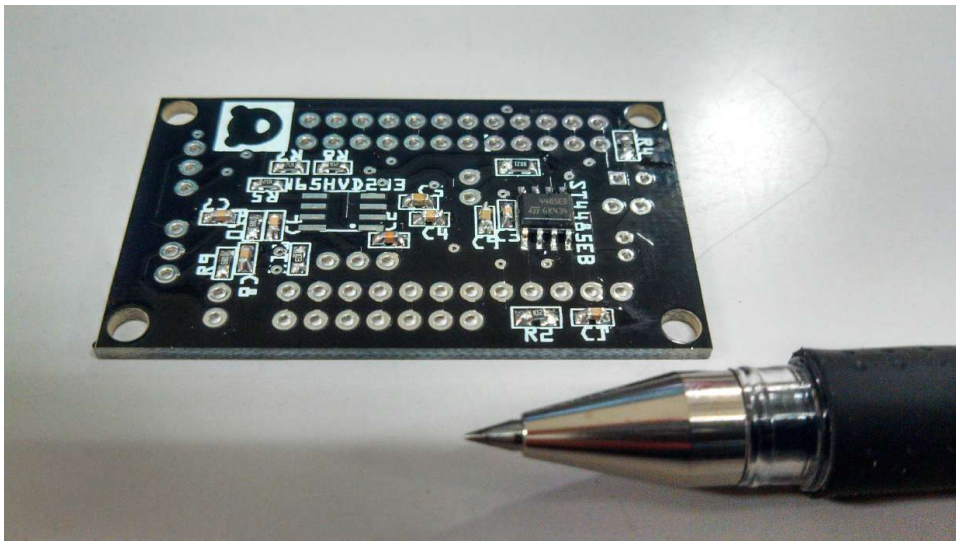
The output voltage V_{out} can then be calculated as

$$\mathrm{V}_{out} = \frac{(\mathrm{R1}+\mathrm{R3}+\alpha\mathrm{SP})(\mathrm{R2}+\beta\mathrm{SP})}{\mathrm{R1R3}+\alpha\mathrm{R3SP}+(\mathrm{R1}+\mathrm{R3}+\alpha\mathrm{SP})(\mathrm{R2}+\beta\mathrm{SP})}\mathrm{VIN}, \tag{C.4}$$

where SP represents the nominal resistance of the SoftPot, and $\beta = [0, 1]$ determines the SoftPot's position, as seen in Fig. C.4(a). The coefficients α and β are related through the relation $\alpha = 1 - \beta$.



(a)



(b)

FIGURE C.3: **RS-485 Shield**
(a) EAGLE Board layout of the design. (b) Manufactured board

The new set of equivalent resistors $\mathrm{RA} = \mathrm{RA}(\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{SP}, \beta)$ and $\mathrm{RB} = \mathrm{RB}(\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{SP}, \beta)$ determine a new circuit.

As it can be seen in (C.4), the new voltage divider is non-linear w.r.t. the position of the SoftPot β, and the resulting SoftPot's value V_{out} would need the calculation of (C.4) on-line. However, if a weak pull-up resistor is used, the behavior is almost linear, as it can be seen in Fig. C.5, where the pull-up resistor's value between a strong and weak pull-up resistor is compared, while keeping R1 and SP fixed. Fig. C.5(a) and Fig. C.5(b) show the output response V_{out} of the scaling circuit using a strong pull-up resistor (R3=10[k]Ω]) and a weak pull-up resistor (R3=100[k]Ω]), respectively. Also notice than the selection of R2 changes the slope of the response. The prototype used in this thesis uses the parameters listed in C.1. Both Figs. Fig. C.5(a) and (b) show a comparison

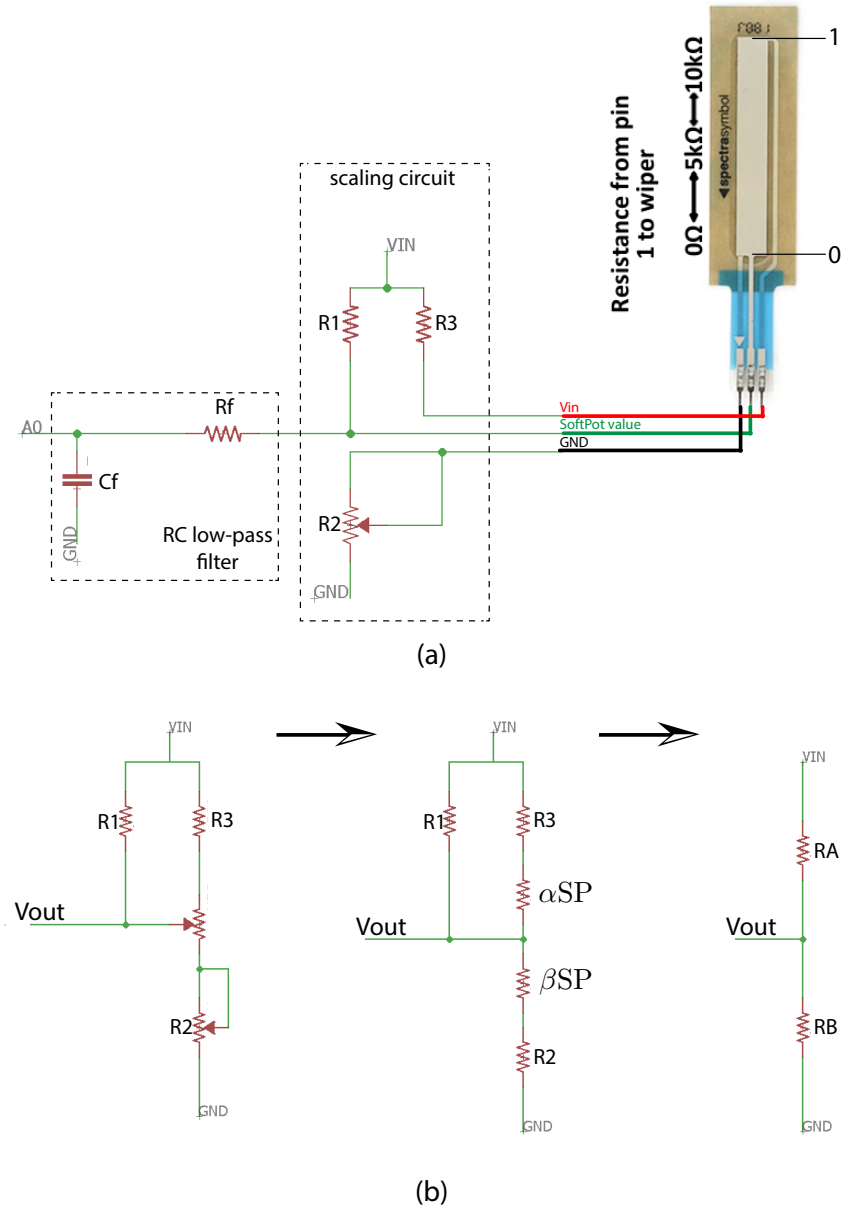
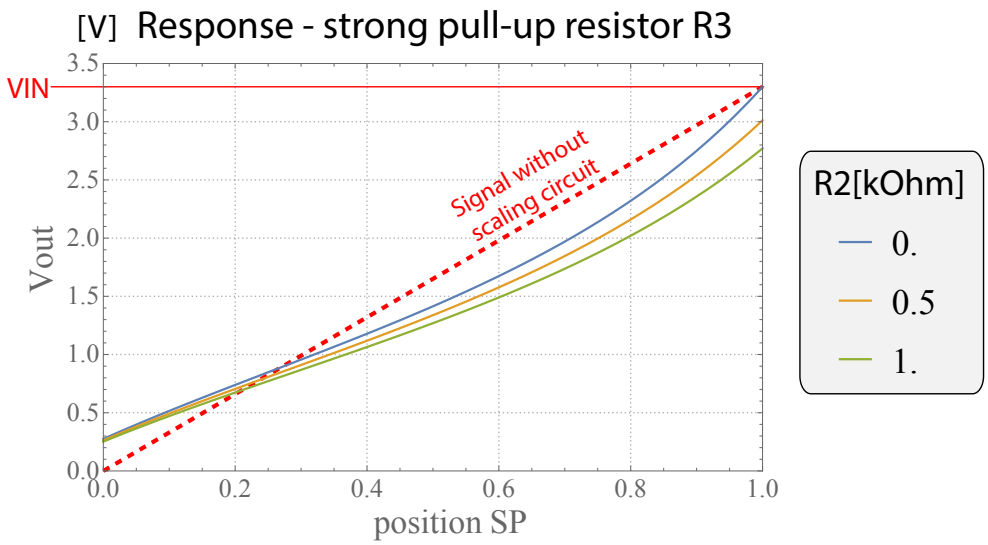


FIGURE C.4: **SoftPot scaling circuit**
(a) Connecting the SoftPot raw signal to a scaling circuit and then to a low-pass filter (b) Analysis and simplification of the scaling circuit

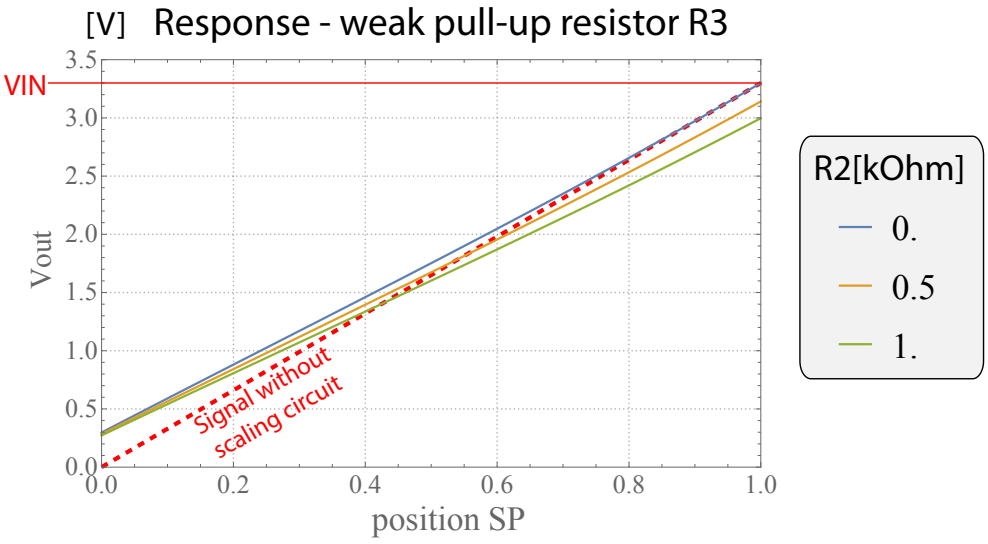
with the ideal original response without the scaling circuit.

TABLE C.1: Parameters of SoftPot Circuit

Parameter	Description	Value	Unit
VIN	input voltage	3.3	[V]
R1	series resistor	1	[kΩ]
R2	series resistor	1	[kΩ]
R3	pull-up resistor	100	[kΩ]
SP	nominal resistance of SoftPot	10	[kΩ]



(a)



(b)

FIGURE C.5: **SoftPot Calibration (VIN=3.3[V],R1=1[kΩ],SP=10[kΩ])** Output response of the scaling circuit by changing the values of R2 and R3 (a) Using a strong pull-up resistor (b) Using a weak pull-up resistor

Appendix D

Programming

D.1 Servo communication - DuoDMXL

DuoDMXL is a library for controlling Dynamixel servos using a Duo (or Photon) as main microcontroller. This library allows the user to write or read all of the possible registers in the servomotors. Specifically, it is meant for MX-64 servomotors. Other models like AX have different EEPROM registers, but the library can be adapted easily.

Communication is delegated to the basic functions:

1. `sendWord()` which writes a value to a register of the servo.
2. `readWord()` which reads the current value of a register of the servo.
3. `readInformation()` which reads a response (set of bytes) from the servomotors. It could be requested information or a simple no-error response.

and all other functions call these basic functions with the appropiate parameters. This not only decreases the code size, but also allows for easier creation of user-defined functions.

The Dynamixel servos have their own protocol for communication, which you can check in the communication section of the ROBOTIS manual `http://support.robotis.com/en/`. Depending on the servos you are communicating with, you need to transform the half-duplex UART signal of the Duo (or Photon) unto a TTL signal for some servos (e.g., MX-64T) which can be done with a tri-state buffer, or use a RS-485 transceiver for servos that use RS-485 bus (e.g., MX-64AR).

The DuoDMXL repository includes eagle schematics and board layouts for two types of 'shields'. The **Duo Tri-state Buffer Shield** is used for half-duplex communication with TTL levels. It takes the TX and RX signal from the DUO (or Photon) and the signal of a control pin, and changes it into communication with only one line of data. The **Duo RS-485 Shield** is used to communicate with a RS-485 transceiver. It takes the TX and RX signal from the DUO (or Photon) and the signal of a control pin and outputs differential communication through the two signals D+ and D- (also called A and B). The library works equally with both hardware setups.

Prerequisites:

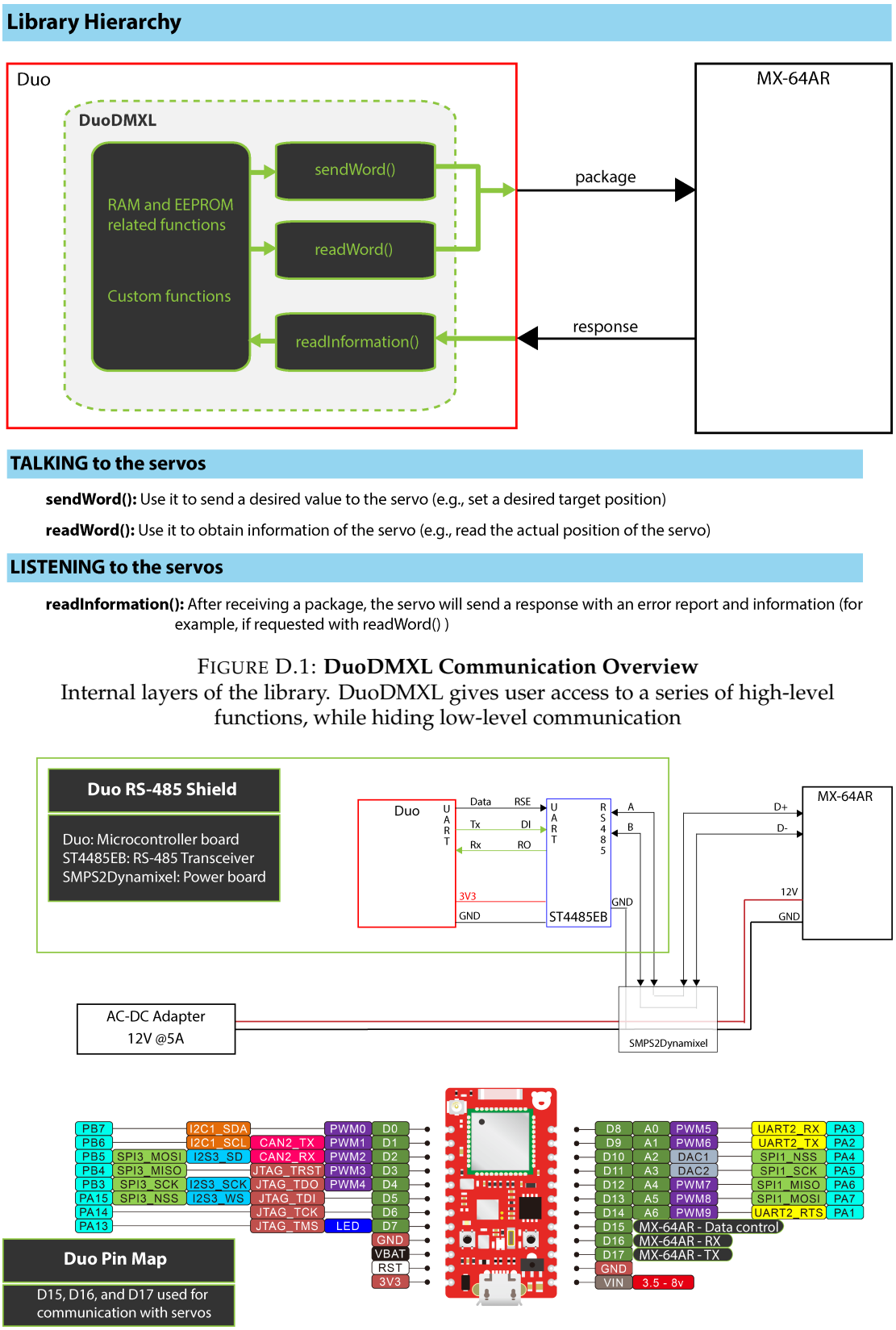


FIGURE D.1: **DuoDMXL Communication Overview**
Internal layers of the library. DuoDMXL gives user access to a series of high-level functions, while hiding low-level communication

FIGURE D.2: **Electronic Setup of Duo and servomotors**
The pins used for communication with the servos, along an overview of the electrical connections

1. Hardware:
    - RS-485 transceiver or Tri-state buffer
2. Software dependencies:
    - None (as of version 2.0)

Installation: This library is mainly intended to be used with the Arduino IDE. Just download the source code in `DuoDMXL_src\` directory and move it to a new directory `...\documents\Arduino\libraries\DuoDMXL\`. The next time you restart the Arduino IDE, the library DuoDMXL will be detected automatically.

D.1.1 Application Programming Interface (API) and Code of DuoDMXL

DuoDMXL provides an API to the user, allowing for full control of the servos. The documentation can be found in the resources listed at Appendix E.1. The main files of DuoDMXL are: `DuoDMXL.h` and `DuoDMXL.cpp` described in Listing E.2 and Listing E.3, respectively, both located in Appendix E.2.1.

Appendix E

Resources

Up-to-date information, schematics, and documentations can be found in the following public repositories.

E.1 Online resources

E.1.1 OPPAS

1. Repository: `https://github.com/FabReyesMecha/OPPAS`
2. Documentation: `https://fabreyesmecha.github.io/OPPAS/`
3. GrabCad: `https://grabcad.com/library/oppas-open-source-parametric-snake-robot-1`

E.1.2 DuoDMXL - Servo Library and Accessories

1. Repository: `https://github.com/FabReyesMecha/DuoDMXL`
2. Documentation: `https://fabreyesmecha.github.io/DuoDMXL/`

E.2 Code

E.2.1 DuoDMXL - Servo Library and Accessories

Listing E.1: DuoDMXL API

Basic functions

This set of functions are intended to work **behind the scenes**.
In other words, the user should not have to worry about them.
They deal with low-level communication and are called by other functions.

`int DynamixelClass::readInformation(void)`

Information |
:--------- |:----------
Description | General function to read the status package from a servo. It then extracts the error (if any) or desired data (for example, if the current positon of a servo was requested)
Notation | **Private** function. Not available to the user
Parameters | None
Returns | A number representing an error or data

`int DynamixelClass::sendWord(uint8_t ID, uint8_t address, int param, int noParams)`

Information |
:--------- |:----------
Description | Function to set the value of a servo's address. noParams should be ONE_BYTE or TWO_BYTES, depending on how many bytes we need to send
Notation | Dynamixel.sendWord(ID, address, param, noParams)
Parameters | uint8_t ID: ID of the servo
| uint8_t address: Address to write to.
| int param: Value to write to the servo's address
| int noParams: Number of bytes to write (one or two bytes)
Returns | A number representing an error (if any)

`int DynamixelClass::readWord(uint8_t ID, uint8_t address, int noParams)`

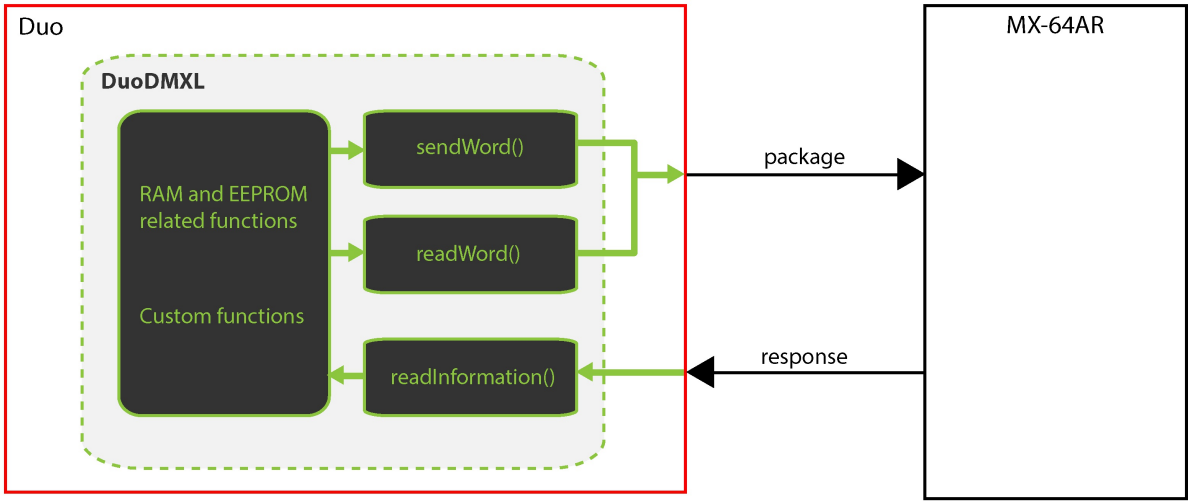
Information |
:--------- |:----------
Description | Function to read the value of a servo's address. noParams should be ONE_BYTE or TWO_BYTES, depending on how many bytes we need
Notation | Dynamixel.readWord(ID, address, noParams)
Parameters | uint8_t ID: ID of the servo
| uint8_t address: Address to read from.
| int noParams: Number of bytes to read (one or two bytes)
Returns | A number representing an error (if any) or the desired data

For example, to change the ID of a servo, a new ID should be written to the register's address 0x03.
This can be accomplished quickly and cleanly with the following code:

```
int DynamixelClass::setID(uint8_t ID, uint8_t newID){
    return(sendWord(ID, EEPROM_ID, newID, ONE_BYTE));
}
```

This should make it easier for the user to write custom functions. The hierarchy of the library is shown (simplified) in the following picture.

Library Hierarchy



TALKING to the servos

sendWord(): Use it to send a desired value to the servo (e.g., set a desired target position)

readWord(): Use it to obtain information of the servo (e.g., read the actual position of the servo)

LISTENING to the servos

readInformation(): After receiving a package, the servo will send a response with an error report and information (for example, if requested with readWord())

Functions related to the EEPROM area of the servos

```
void DynamixelClass::begin(long baud, uint8_t directionPin)
```

Information |
:--------- |:----------
Description | Initialize communication w ith the servos w ith a user-defined pin for the data direction control
Notation | Dynamixel.begin(baud, directionPin)
Parameters | long baud: Desired baudrate for communication w ith the servos. For MX-64AR the default baudrate is 1Mbps (if I remember correctly)
| uint8_t directionPin: direction used for flow control.
Returns | Nothing

```
void DynamixelClass::begin(long baud)
```

Information |
:--------- |:----------
Description | Initialize communication w ith the servos w ith a pre-defined pin (D15) for the data direction control
Notation | Dynamixel.begin(baud)
Parameters | long baud: Desired baudrate for communication w ith the servos.
Returns | Nothing

```
void DynamixelClass::end()
```

Information |
:--------- |:----------
Description | End communication w ith the servos
Notation | Dynamixel.end()
Parameters | None

Returns | Nothing

`int DynamixelClass::readModel(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to read the servo model. EEPROM Address 0(x00) and 1(0x01)
Notation | model = Dynamixel.readModel(ID)
Parameters | uint8_t ID: ID of the servo
Returns | ID of the servo

`int DynamixelClass::readFirmware(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to read the version of the firmware. EEPROM Address 2(0x02)
Notation | fw = Dynamixel.readFirmware(ID)
Parameters | uint8_t ID: ID of the servo
Returns | Number representing the firmware version of the servo

`int DynamixelClass::setID(uint8_t ID, uint8_t newID)`

Information |
:--------- |:----------
Description | Function to set the ID of the servo. EEPROM Address 3(0x03)
Notation | Dynamixel.setID(ID, new ID)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t new ID: New ID of the servomotor
Returns | Number representing an error (if any)

`int DynamixelClass::readID(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to read the ID of the servo. EEPROM Address 3(0x03)
Notation | id = Dynamixel.readID(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Current ID of the servo

`int DynamixelClass::setBD(uint8_t ID, long baudrate)`

Information |
:--------- |:----------
Description | Function to set baudrate. EEPROM Address 4(0x04)
Notation | Dynamixel.setBD(ID, baudrate)
Parameters | uint8_t ID: Current ID of the servomotor
| long baudrate: Desired baudrate for communication. Up to 1Mbps (use 1000000) is officially supported by the servos. This functions truncates the number so there may be a small error on the final value. If you need a precies value, use `setBDTable(ID, baud)` where the following relationship holds: `baud = (2000000/baudrate) -1`
Returns | Number representing an error (if any)

NOTE: After changing the baudrate you need to restart the serial communication with the servos, using the new baudrate. For example:

```
//begin communication with the current baudrate
Dynamixel.begin(baudrate, dataPin);
```

```
      //change baudrate
      Dynamixel.setBD(servoID, newBaudrate);

      //stop communication and restart with the new baudrate
      Dynamixe.end();
      Dynamixel.begin(newBaudrate, datapin);
```

`int DynamixelClass::setBDTable(uint8_t ID, uint8_t baud)`

Information |
:--------- |:----------
Description | Function to set baudrate based on the manual's table. EEPROM Address 4(0x04). This is a more precise value. For example, for communication at 57600 bps use '34'.
Notation | Dynamixel.setBDTable(ID, baud)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t baud: Number from 0 to 255 representing a desired baudrate for communication. The resulting baudrate in bps can be calculated as: `baudrate = 2000000/(baud + 1)`
Returns | Number representing an error (if any)

NOTE: After changing the baudrate you need to restart the serial communication with the servos, using the new baudrate. See the previous note and following example:

```
      unsigned long baudrate = 57600             //Currently, the servos are communicating a 57600 bps
      uint8_t newBaud = 9;                       //A value of 9 corresponds to 200,000 bps. Check ROBOTIS documentation

      //begin communication with the current baudrate
      Dynamixel.begin(baudrate, dataPin);

      //change baudrate
      Dynamixel.setBDTable(servoID, newBaud);

      //stop communication and restart with the new baudrate
      Dynamixe.end();

      unsigned long baudrate = 2000000/(newBaud+1);   //This transforms the value 9 into 200,000

      Dynamixel.begin(baudrate, datapin);
```

`int DynamixelClass::readBD(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to read the setting of the baudrate. EEPROM Address 4(0x04)
Notation | baudrate = Dynamixel.readBD(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Number from 0 to 255 representing the baudrate currently being used

`int DynamixelClass::setRDT(uint8_t ID, uint8_t RDT)`

Information |
:--------- |:----------
Description | Set the Return Delay Time (RDT) in microseconds. EEPROM Address 5(0x05)
Notation | Dynamixel.setRDT(ID, RDT)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t RDT: Desired RDT value
Returns | Number representing an error (if any)

`int DynamixelClass::readRDT(uint8_t ID)`

Information |
:--------- |:----------
Description | Read the Return Delay Time (RDT) value. EEPROM Address 5(0x05)
Notation | rdt = Dynamixel.readRDT(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Current value of RDT

```
int DynamixelClass::setCWAngleLimit(uint8_t ID, int limit)
```

Information |
:--------- |:----------
Description | Set the value for the CW Angle limit. EEPROM Address 6(0x06) and 7(0x07)
Notation | Dynamixel.setCWAngleLimit(ID, limit)
Parameters | uint8_t ID: Current ID of the servomotor
| int limit: Desired limit
Returns | Number representing an error (if any)

```
int DynamixelClass::readCWAngleLimit(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the value for the CW Angle limit. EEPROM Address 6(0x06) and 7(0x07)
Notation | cw Limit = Dynamixel.readCWAngleLimit(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Angle being used for clockw ise limit of the servo

```
int DynamixelClass::setCCWAngleLimit(uint8_t ID, int limit)
```

Information |
:--------- |:----------
Description | Set the value for the CCW Angle limit. EEPROM Address 8(0x08) and 9(0x09)
Notation | Dynamixel.setCCWAngleLimit(ID, limit)
Parameters | uint8_t ID: Current ID of the servomotor
| int limit: Desired limit
Returns | Number representing an error (if any)

```
int DynamixelClass::readCCWAngleLimit(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the value for the CCW Angle limit. EEPROM Address 8(0x08) and 9(0x09)
Notation | ccw Limit = Dynamixel.readCCWAngleLimit(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Angle being used for counter-clockw ise limit of the servo

```
int DynamixelClass::setTempLimit(uint8_t ID, uint8_t Temperature)
```

Information |
:--------- |:----------
Description | Set the limit temperature. EEPROM Address 11(0x0B)
Notation | Dynamixel.setTempLimit(ID, Temperature)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t Temperature: Temperature that w ill be set as limit. The servo w ill shutdow n if this temperature is reached. Documentation suggests not modifying the default value.
Returns | Number representing an error (if any)

```
int DynamixelClass::readTempLimit(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the limit temperature. EEPROM Address 11(0x0B)
Notation | tempLimit = Dynamixel.readTempLimit(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Temperature used as upper limit

```
int DynamixelClass::setLowVoltageLimit(uint8_t ID, uint8_t lowVoltage)
```

Information |
:--------- |:----------
Description | Set the low est voltage limit. EEPROM Address 12(0x0C)
Notation | Dynamixel.setLow VoltageLimit(ID, low Voltage)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t low Voltage: Low er bound for voltage limit
Returns | Number representing an error (if any)

```
int DynamixelClass::readLowVoltageLimit(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the low est voltage limit. EEPROM Address 12(0x0C)
Notation | low VoltageLimit = Dynamixel.readLow VoltageLimit(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Low er bound voltage

```
int DynamixelClass::setHighVoltageLimit(uint8_t ID, uint8_t highVoltage)
```

Information |
:--------- |:----------
Description | Set the highest voltage limit. EEPROM Address 13(0x0D)
Notation | Dynamixel.setHighVoltageLimit(ID, highVoltage)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t highVoltage: Upper bound for voltage limit
Returns | Number representing an error (if any)

```
int DynamixelClass::readHighVoltageLimit(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the highest voltage limit. EEPROM Address 13(0x0D)
Notation | highVoltageLimit = Dynamixel.readHighVoltageLimit(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Upper bound voltage

```
int DynamixelClass::setMaxTorque(uint8_t ID, int MaxTorque)
```

Information |
:--------- |:----------
Description | Set the maximum torque. EEPROM Address 14(0x0E) and 15(0x0F)
Notation | Dynamixel.setMaxTorque(ID, MaxTorque)
Parameters | uint8_t ID: Current ID of the servomotor
| int MaxTorque: Value from [0, 1023] used for maximum output torque.
Returns | Number representing an error (if any)

```
int DynamixelClass::readMaxTorque(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the maximum torque. EEPROM Address 14(0x0E) and 15(0x0F)
Notation | maxTorque = Dynamixel.readMaxTorque(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Value from [0, 1023] representing the maximum output torque currently used.

`int DynamixelClass::setSRL(uint8_t ID, uint8_t SRL)`

Improved functionality in **DuoDMXL** v.0.3.

Information |
:--------- |:----------
Description | Set the Status Return Level. EEPROM Address 16(0x10)
Notation | Dynamixel.setSRL(ID, SRL)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t SRL: 0, 1, or 2 for 'no return against all commands', 'return only for the READ command', or 'Return for all commands', respectively.
Returns | Number representing an error (if any)

As of **DuoDMXL** v.0.3 the user can choose any SRL.
How ever, **DuoDMXL** assumes on reset that all servos have the same SRL and ALL communications return a package (i.e., `SRL=2`).
If you changed the value of SRL on a previous session, there may be problems w ith the communication, since the value of SRL is w ritten in the EEPROM memory of the servos.
If you are having troubles w ith communication use:

```
Dynamixel.begin(baud, dataPin);
delay(500);
Dynamixel.setSRL(254, 2);
```

at the beginning of your program to set all servos to 'Return for all commands'.

NOTE: `254` is the broadcast ID. Any command is sent to all servos.

`int DynamixelClass::readSRL(uint8_t ID)`

Information |
:--------- |:----------
Description | Read the Status Return Level value. EEPROM Address 16(0x10)
Notation | srl = Dynamixek.readSRL(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Actual value of the SRL address

`int DynamixelClass::setAlarmLED(uint8_t ID, uint8_t alarm)`

Information |
:--------- |:----------
Description | Set Alarm LED. EEPROM Address 17(0x11)
Notation | Dynamixel.setAlarmLED(ID, alarm)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t alarm: The servo's LED w ill blink if an error occurs
Returns | Number representing an error (if any)

`int DynamixelClass::readAlarmLED(uint8_t ID)`

Information |
:--------- |:----------
Description | Read Alarm LED value. EEPROM Address 17(0x11)
Notation | alarm = Dynamixel.readAlarmLED(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Actual value of the alarm address

```
int DynamixelClass::setShutdownAlarm(uint8_t ID, uint8_t SALARM)
```

Information |
:--------- |:----------
Description | Set Shutdow n alarm. EEPROM Address 18(0x12)
Notation | Dynamixel.setShutdow nAlarm(ID, SALARM)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t SALARM: Check the documentation. Depending on the value sent, the servo w ill output a 0% torque if an alarm is activated. By default the value is 36 (0x24) w hich in binary is '0010 0100', meaning overload+overheating error.
Returns | Number representing an error (if any)

Errors

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	instruction error	overload error	checksum error	range error	overheating error	angle limit error	input voltage error

```
int DynamixelClass::readShutdownAlarm(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read Shutdow n alarm value. EEPROM Address 18(0x12)
Notation | alarm = Dynamixel.readShutdow nAlarm(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Read the value of the alarm address to verify if an alarm w as activated

```
int DynamixelClass::setMultiTurnOffset(uint8_t ID, int offset)
```

Information |
:--------- |:----------
Description | Set the multi-turn offset values. EEPROM ADDRESS: 20(0x14) and 21(0x15)
Notation | Dynamixel.setMultiTurnOffset(ID, offset)
Parameters | uint8_t ID: Current ID of the servomotor
| int offset: Angle used for servo's offset in multi-turn mode
Returns | Number representing an error (if any)

```
int DynamixelClass::readMultiTurnOffset(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the multi-turn offset values. EEPROM ADDRESS: 20(0x14) and 21(0x15)
Notation | offset = Dynamixel.readMultiTurnOffset(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Read the actual offset being used for multi-turn mode

```
int DynamixelClass::setResolutionDivider(uint8_t ID, uint8_t divider)
```

Information |
:--------- |:----------
Description | Set the resolution divider value. EEPROM ADDRESS: 22(0x16)

Notation | Dynamixel.setResolutionDivider(ID, divider)
Parameters | uint8_t ID: Current ID of the servomotor
| uint8_t divider: Divider for the servo's angle. By default divider=1
Returns | Number representing an error (if any)

```
int DynamixelClass::readResolutionDivider(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the resolution divider value. EEPROM ADDRESS: 22(0x16)
Notation | divider = Dynamixel.readResolutionDivider(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Actual value used for the servo's divider

Functions related to the RAM area of the servos

```
int DynamixelClass::torqueEnable( uint8_t ID, bool Status)
```

Information |
:--------- |:----------
Description | Function to turn ON or OFF torque. RAM Address 24(0x18)
Notation | Dynamixel.torqueEnable(ID, Status)
Parameters | uint8_t ID: Current ID of the servomotor
| bool Status: True or False for enabling or disabling torque, respectively.
Returns | Number representing an error (if any)

```
int DynamixelClass::torqueEnableStatus( uint8_t ID)
```

Information |
:--------- |:----------
Description | Function to check if the servo generates torque. RAM Address 24(0x18)
Notation | status = Dynamixel.torqueEnableStatus(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | 1 if torque is enabled. 0 if disabled

```
int DynamixelClass::ledStatus(uint8_t ID, bool Status)
```

Information |
:--------- |:----------
Description | Function to turn ON or OFF the servo's LED. RAM Address 25(0x19)
Notation | Dynamixel.ledStatus(ID, Status)
Parameters | uint8_t ID: Current ID of the servomotor
| bool Status: True or False for enabling or disabling the LED, respectively. I do not provide a function to read this register, so **visual confirmation** should be used.
Returns | Number representing an error (if any)

```
int DynamixelClass::setGainD(uint8_t ID, int gain)
```

Information |
:--------- |:----------
Description | Function to set the value of the Derivative gain. RAM Address 26(0x1A)
Notation | Dynamixel.setGainD(ID, gain)
Parameters | uint8_t ID: Current ID of the servomotor
| int gain: Number [0,254] used for the Derivative gain of the servo. Related to the servo's PID control
Returns | Number representing an error (if any)

`int DynamixelClass::readGainD(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to read the value of the Derivative gain. RAM Address 26(0x1A)
Notation | gainD = Dynamixel.readGainD(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Derivative gain

`int DynamixelClass::setGainI(uint8_t ID, int gain)`

Information |
:--------- |:----------
Description | Function to set the value of the Integral gain. RAM Address 27(0x1B)
Notation | Dynamixel.setGainI(ID, gain)
Parameters | uint8_t ID: Current ID of the servomotor
| int gain: Number [0,254] used for the Integral gain of the servo. Related to the servo's PID control
Returns | Number representing an error (if any)

`int DynamixelClass::readGainI(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to read the value of the Integral gain. RAM Address 27(0x1B)
Notation | gainI = Dynamixel.readGainI(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Integral gain

`int DynamixelClass::setGainP(uint8_t ID, int gain)`

Information |
:--------- |:----------
Description | Function to set the value of the Proportional gain. RAM Address 28(0x1C)
Notation | Dynamixel.setGainP(ID, gain)
Parameters | uint8_t ID: Current ID of the servomotor
| int gain: Number [0,254] used for the Proportional gain of the servo. Related to the servo's PID control
Returns | Number representing an error (if any)

`int DynamixelClass::readGainP(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to read the value of the Proportional gain. RAM Address 28(0x1C)
Notation | gainP = Dynamixel.readGainP(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Proportional gain

`int DynamixelClass::move(uint8_t ID, int Position)`

Information |
:--------- |:----------
Description | Function to move servo to a specific position. RAM Address 30(0x1E) and 31(0x1F)
Notation | Dynamixel.move(ID, Position)
Parameters | uint8_t ID: Current ID of the servomotor
| int Position: Number [0,4095] representing the desired position. The unit is about 0.088 degrees
Returns | Number representing an error (if any)

`int DynamixelClass::setMovingSpeed(uint8_t ID, int speed)`

Information |
:--------- |:----------
Description | Function to set the desired moving speed. RAM Address 32(0x20) and 33(0x21)
Notation | Dynamixel.setMovingSpeed(ID, speed)
Parameters | uint8_t ID: Current ID of the servomotor
| int speed: Number [0,1023] representing moving speed. The unit is about 0.114 rpm.
Returns | Number representing an error (if any)

`int DynamixelClass::readMovingSpeed(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to read the desired moving speed. RAM Address 32(0x20) and 33(0x21)
Notation | speed = Dynamixel.readMovingSpeed(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Moving speed value in the register

`int DynamixelClass::setTorqueLimit(uint8_t ID, int torque)`

Information |
:--------- |:----------
Description | Function to set the value of the goal torque. RAM Address 34(0x22) and 35(0x23)
Notation | Dynamixel.setTorqueLimit(ID, torque)
Parameters | uint8_t ID: Current ID of the servomotor
| int torque: Number [0,1023] used for torque limit. The servo will not exert a higher torque. According to documentation, if the motor is disabled due to activating an alarm, this register will be set to 0. To re-enable the servo set a non-zero value. When the motor is turned on, the torque limit will take the value currently written in the EEPROM address 0x0E. Check the function **setMaxTorque()** if you want to change this value.
Returns | Number representing an error (if any)

`int DynamixelClass::readTorqueLimit(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to read the value of the goal torque. RAM Address 34(0x22) and 35(0x23)
Notation | torqueLimit = readTorqueLimit(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Torque limit value [0,1023]

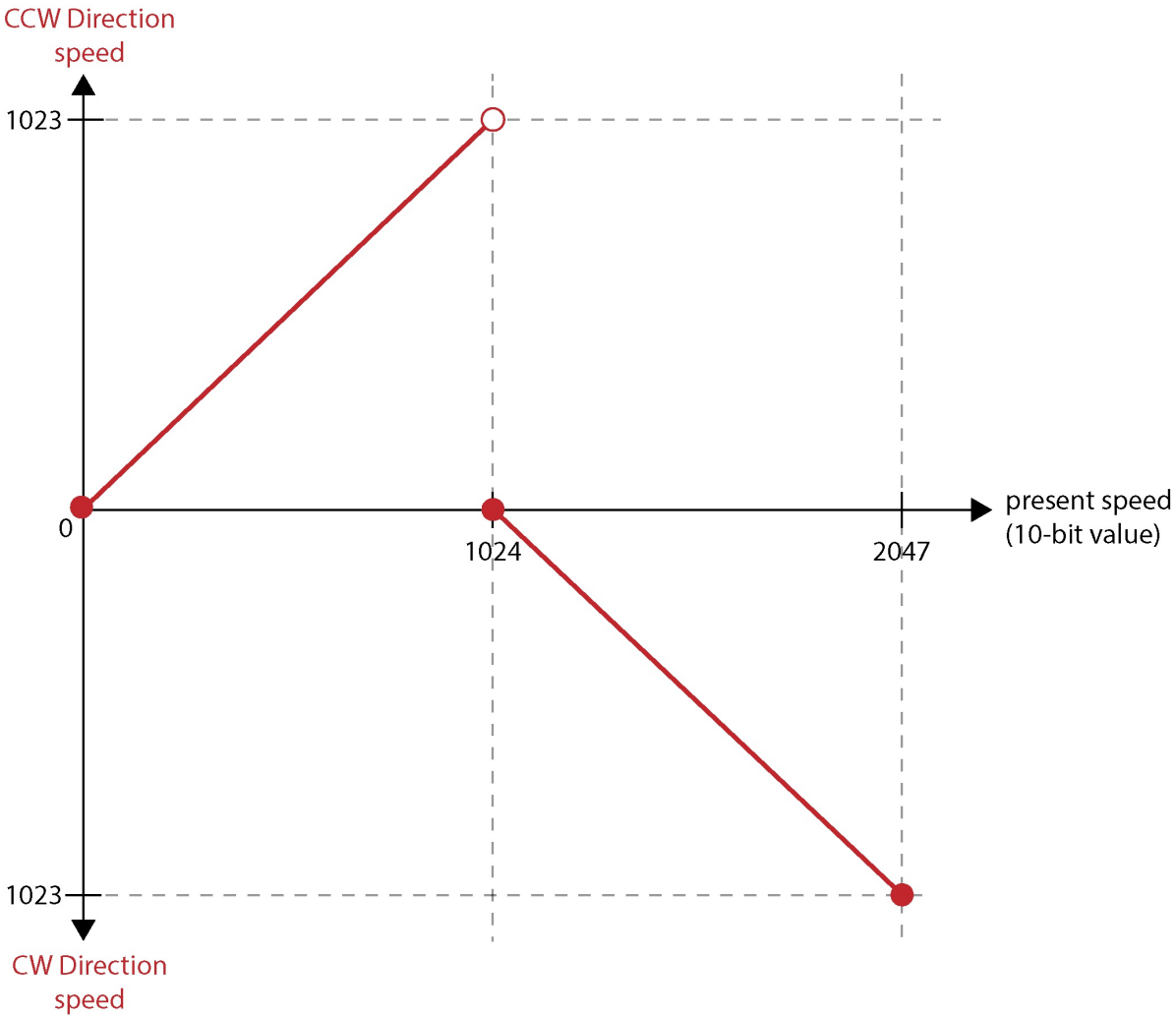
`int DynamixelClass::readPosition(uint8_t ID)`

Information |
:--------- |:----------
Description | Read the actual position. RAM Address 36(0x24) and 37(0x25)
Notation | pos = Dynamixel.readPosition(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | The actual position of the servo

`int DynamixelClass::readSpeed(uint8_t ID)`

Information |
:--------- |:----------
Description | Read the actual speed. RAM Address 38(0x26) and 39(0x27)
Notation | speed = Dynamixel.readSpeed(ID)
Parameters | uint8_t ID: Current ID of the servomotor

Returns | The actual speed of the servo (possible values [0,2047]). [0,1023] correspond to CCW and [1024,2047] to CW speeds, respectively. In other w ords, the magnitude is given by the first nine bits [0,1023] and the tenth bit denotes the direction.




```
int DynamixelClass::readLoad(uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the load. RAM Address 40(0x28) and 41(0x29)
Notation | load = Dynamixel.readLoad(ID)
Parameters | uint8_t ID: Current ID of the servomotor
| Reads the currently applied load. How ever, according to documentation, this value is inferred from the internal torque value and should not be used for accurate torque measurement. It is better to read the current.
Returns | The currently applied load. If the load is CCW the value w ill be [0,1023] and of CW then [1024,2047]

```
int DynamixelClass::readVoltage(uint8_t ID)
```

Information |
:--------- |:----------
Description | Function to read the voltage. RAM Address 42(0x2A)
Notation | voltage = Dynamixel.readVoltage(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | (10 times) The actual voltage

```
int DynamixelClass::readTemperature(uint8_t ID)
```

Information |
:--------- |:----------
Description | Function to read the Temperature. RAM Address 43(0x2B)
Notation | temp = Dynamixel.readTemperature(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Actual temperature of the servo (Celsius degrees)

`int DynamixelClass::registeredStatus(uint8_t ID)`

Information |
:--------- |:----------
Description | Check if there is an instruction registered. RAM Address 44(0x2C)
Notation | status = Dyunamixel.registeredStatus(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | 0 if there are no commands registered, and 1 otherwise

`int DynamixelClass::moving(uint8_t ID)`

Information |
:--------- |:----------
Description | Check if goal position command is being executed (Address 0x30?). RAM Address 46(0x2E)
Notation | moving = Dynamixel.moving(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | 0 if the servo is not moving and 1 if otherwise

`int DynamixelClass::lockEEPROM(uint8_t ID)`

Information |
:--------- |:----------
Description | Locks the EEPROM area. RAM Address 47(0x2F). Power must be turned off to reset it
Notation | Dynamixel.lockEEPROM(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Number representing an error (if any)

`int DynamixelClass::setPunch(uint8_t ID, int Punch)`

Information |
:--------- |:----------
Description | RAM Address 48(0x30) and 49(0x31). Honestly, I do not understand this feature even after reading the documentation
Notation | Dynamixel.setPunch(ID, Punch)
Parameters | uint8_t ID: Current ID of the servomotor
| int Punch: number [0,1023]
Returns | Number representing an error (if any)

`int DynamixelClass::readPunch(uint8_t ID)`

Information |
:--------- |:----------
Description | Reads the value of RAM Address 48(0x30) and 49(0x31). See the note in **setPunch()**
Notation | punch = Dynamixel.readPunch(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | punch

`int DynamixelClass::readCurrent(uint8_t ID)`

Information |

:--------- |:----------
Description | Function to read the current. RAM ADDRESS: 68(0x44) and 69(0x45)
Notation | current = Dynamixel.readCurrent(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Actual current running through the servo

TODO: Add picture

```
int DynamixelClass::torqueControl( uint8_t ID, bool enable)
```

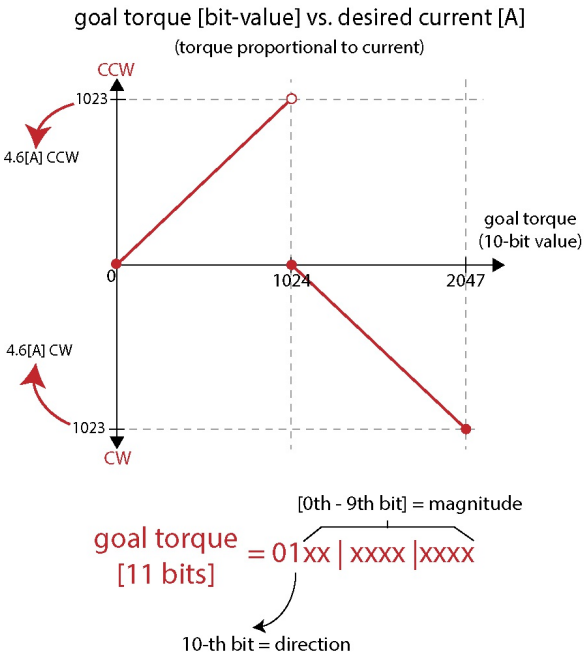
Information |
:--------- |:----------
Description | Torque control mode enable. RAM ADDRESS: 70(0x46). The servo can run continuously trying to achieve the desired (goal) torque. See **setGoalTorque()**. When torque mode is enabled, you can no longer control the servo's positon.
Notation | Dynamixel.torqueControl(ID, enable)
Parameters | uint8_t ID : Current ID of the servomotor
| bool enable: **True** for enabling and **False** for disabling
Returns | Number representing an error (if any)

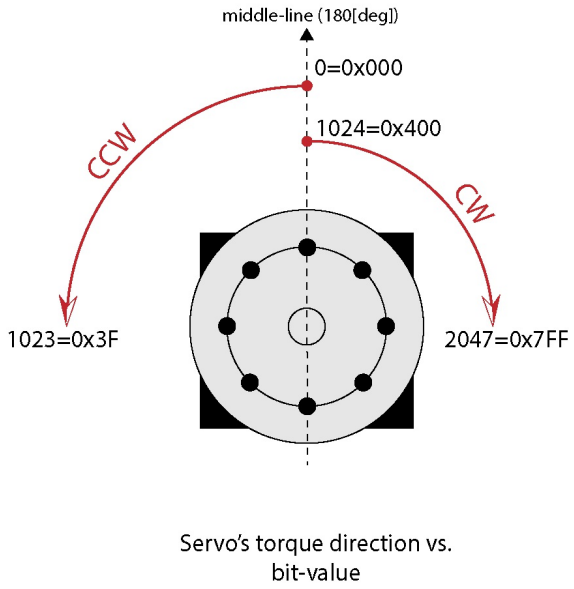
```
int DynamixelClass::readTorqueControl( uint8_t ID)
```

Information |
:--------- |:----------
Description | Read the Torque control mode status. RAM ADDRESS: 70(0x46)
Notation | status = Dynamixel.readTorqueControl(ID)
Parameters | uint8_t ID : Current ID of the servomotor
Returns | 1 if torque mode is enabled and 0 otherwise

```
int DynamixelClass::setGoalTorque(uint8_t ID, int torque)
```

Information |
:--------- |:----------
Description | Function to set the goal torque. RAM ADDRESS: 71(0x47) and 72(0x48). In reality you are setting a desired current and the torque will be proportional
Notation | Dynamixel.setGoalTorque(ID, torque)
Parameters | uint8_t ID : Current ID of the servomotor
| int torque: [0,1023] for CCW torque and [1024,2047] for CW. The unit is 4.5 [mA]. The goal torque value cannot be bigger than the torque limit value
Returns | Number representing an error (if any)





Servo's torque direction vs.
bit-value

```
int DynamixelClass::setGoalAccel(uint8_t ID, uint8_t accel)
```

Information |
:--------- |:----------
Description | Function to set goal acceleration/ RAM ADDRESS: 73(0x49)
Notation | Dynamixel.setGoalAccel(ID, accel)
Parameters | uint8_t ID : Current ID of the servomotor
| uint8_t accel: desired acceleration [0,254]
Returns | Number representing an error (if any)

Custom Functions

These functions build upon the previous ones to provide more advanced functionality.
In particular there are several functions in order to find your connected servo in case you don't know its ID or the baudrate it is communicating with.
This is very useful for debugging new code that may overwrite a register accidentaly.
For example, if you can' remember your servo's ID or don't know the baudrate it is using, and you do not have the USB2Dynamixel product to debug your servo, then you can use this functions to find it.
Needless to say, I wrote these functions because at some point I accidentally changed the servo's ID and baudrate to an unknown value and couldn't communicate with it.
Connect only one servo to find it.

```
void DynamixelClass::configureServo(uint8_t ID, uint8_t newID, long baud)
```

Information |
:--------- |:----------
Description | Configure both ID and Baudrate of the servo
Notation | Dynamixel.configureServo(ID, new ID, baud)
Parameters | uint8_t ID : Current ID of the servomotor
| uint8_t new ID : New ID for the servomotor
| long baud : New baudrate for communication
Returns | Nothing

```
void DynamixelClass::setAngleLimit(uint8_t ID, int CWLimit, int CCWLimit)
```

Information |
:--------- |:----------
Description | Set both angle limits
Notation | Dynamixel.setAngleLimit(ID, CWLimit, CCWLimit)
Parameters | uint8_t ID : Current ID of the servomotor
| int CWLimit: Clockw ise limit for the servo
| int CCWLimit: Counter-clockw ise limit for the servo
Returns | Nothing

`void DynamixelClass::setWheelMode(uint8_t ID, bool enable)`

Information |
:--------- |:----------
Description | Function to set both limits to 0. The servo is functioning in w heel mode
Notation | Dynamixel.setWheelMode(ID, enable)
Parameters | uint8_t ID: Current ID of the servomotor
| bool enable: Enables or disables servo's w heel mode. If disabled, the servo defaults to the usual [0,4095] range of motion
Returns | Nothing

`void DynamixelClass::setJointMode(uint8_t ID)`

Information |
:--------- |:----------
Description | Function to set the servo as joint mode. Equivalent to setWheelMode(ID, false)
Notation | Dynamixel.setJointMode(ID)
Parameters | uint8_t ID: Current ID of the servomotor
Returns | Nothing

`void DynamixelClass::setDIP(uint8_t ID, int gainD, int gainI, int gainP)`

Information |
:--------- |:----------
Description | Function to set all gains. Be careful w ith the order
Notation | Dynamixel.setDIP(ID, gainD, gainI, gainP)
Parameters | uint8_t ID : Current ID of the servomotor
| int gainD: Derivative gain
| int gainI: Integral gain
| int gainP: Proportional gain
Returns | Nothing

`int DynamixelClass::findByBaudRate(long baudRate)`

Information |
:--------- |:----------
Description | Function to find the ID of the servo, if you have the correct baudrate. Assume begin() has been called
Notation | foundID = Dynamixel.findByBaudRate(baudRate)
Parameters | long baudRate: baudrate used for communication
Returns | Found ID of the servo. If no servo is found (meaning you have the incorrect baudrate) then the value returned is -1

`int DynamixelClass::findByID(uint8_t id, uint8_t directionPin)`

Information |
:--------- |:----------
Description | Function to find the baudrate to communicate w ith the servo, if you have the correct ID. Assume begin() has NOT been called
Notation | Dynamixel.findByID(id, directionPin)
Parameters | uint8_t id: Current id of the servomotor. This function assumes the ID is correct and you are only trying to find the

correct baudrate
| uint8_t directionPin: The pin used for data flow control.
Returns | The baudrate (table value [0,255]) representing the baudrate. Returns -1 if the servo was not found

`void DynamixelClass::findServo(uint8_t directionPin)`

Information |
:--------- |:----------
Description | Find the servo without having any information. Assume begin() has NOT been called. The function doesn't return anything but prints the result to the terminal (Use the Arduino IDE built in terminal monitor). This is the last resort to find your servo info. Since it tries all baudrates and IDs, it may take a while
Notation | Dynamixel.findServo(directionPin)
Parameters | uint8_t directionPin: Pin used for flow control
Returns | Nothing. Check the output in the terminal

`void DynamixelClass::changeTimeOut(uint8_t newTimeOut)`

Introduced in **DuoDMXL** v.0.3.

Information |
:--------- |:----------
Description | Change time out period (waiting time for status package). Unit is [ms]. The maximum value is 255. The default value is 50[ms]
Notation | Dynamixel.changeTimeOut(newTimeOut)
Parameters | uint8_t newTimeOut: New time out period
Returns | Nothing

`void DynamixelClass::changeCoolDown(uint16_t newCoolDown)`

Introduced in **DuoDMXL** v.0.3.

Information |
:--------- |:----------
Description | Change cool down period (time between sending commands). Unit is [ms]. The maximum value is 65,535 (i.e., 65.535 seconds). The default value is 0[ms]
Notation | Dynamixel.changeCoolDown(newCoolDown)
Parameters | uint16_t newCoolDown: New cool down period
Returns | Nothing

Listing E.2: DuoDMXL.h

```
/*
DuoDMXL v2.0
MX-64AR Half Duplex USART/RS-485 Communication Library
-------------------------------------------------------------------------------

Target Boards:
      Redbear Duo
      Particle Photon (not tested)
      Arduino Leonardo
      or any other board with two hardware serial ports (soft serial not
         tested)

DuoDMXL.h:
      Definitions of constants and methods for class DynamixelClass

      Initially based on Savage's DynamixelSerial Library
      http://savageelectronics.blogspot.jp/2011/01/arduino-y-dynamixel-ax
         -12.html
-------------------------------------------------------------------------------


Copyright(c) 2016 Fabian Eugenio Reyes Pinner
Created by Fabian E. Reyes Pinner on 2016-06-01 (August 1st, 2016)

This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU Lesser General Public License as published
      by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
   GNU Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public License
   along with this program. If not, see <http://www.gnu.org/licenses/>.

-------------------------------------------------------------------------------

 */

#ifndef DuoDMXL_h
#define DuoDMXL_h
```

```
      // EEPROM AREA
         //////////////////////////////////////////////////////
#define EEPROM_MODEL_NUMBER_L     0
#define EEPROM_MODEL_NUMBER_H     1
#define EEPROM_VERSION        2
#define EEPROM_ID           3
#define EEPROM_BAUD_RATE       4
#define EEPROM_RETURN_DELAY_TIME   5
#define EEPROM_CW_ANGLE_LIMIT_L    6
#define EEPROM_CW_ANGLE_LIMIT_H    7
#define EEPROM_CCW_ANGLE_LIMIT_L   8
#define EEPROM_CCW_ANGLE_LIMIT_H   9
#define EEPROM_LIMIT_TEMPERATURE   11
#define EEPROM_DOWN_LIMIT_VOLTAGE   12
#define EEPROM_UP_LIMIT_VOLTAGE    13
#define EEPROM_MAX_TORQUE_L      14
#define EEPROM_MAX_TORQUE_H      15
#define EEPROM_RETURN_LEVEL      16
#define EEPROM_ALARM_LED       17
#define EEPROM_ALARM_SHUTDOWN     18
#define EEPROM_TURN_OFFSET_L     20   //MX-64
#define EEPROM_TURN_OFFSET_H     21   //MX-64
#define EEPROM_RESOLUTION_DIV     22   //MX-64

// RAM AREA /////////////////////////////////////////////////////////
#define RAM_TORQUE_ENABLE       24
#define RAM_LED            25
#define RAM_CW_COM_MAR        26   //AX-12
#define RAM_CCW_COM_MAR        27   //AX-12
#define RAM_CW_COM_SLOPE       28   //AX-12
#define RAM_CCW_COM_SLOPE       29   //AX-12
#define RAM_D_GAIN          26   //MX-64
#define RAM_I_GAIN          27   //MX-64
#define RAM_P_GAIN          28   //MX-64
#define RAM_GOAL_POSITION_L      30
#define RAM_GOAL_POSITION_H      31
#define RAM_GOAL_SPEED_L       32
#define RAM_GOAL_SPEED_H       33
#define RAM_TORQUE_LIMIT_L      34
#define RAM_TORQUE_LIMIT_H      35
#define RAM_PRESENT_POSITION_L    36
#define RAM_PRESENT_POSITION_H    37
#define RAM_PRESENT_SPEED_L      38
#define RAM_PRESENT_SPEED_H      39
#define RAM_PRESENT_LOAD_L      40
#define RAM_PRESENT_LOAD_H      41
#define RAM_PRESENT_VOLTAGE      42
#define RAM_PRESENT_TEMPERATURE    43
```

```
#define RAM_REGISTERED_INSTRUCTION   44
#define RAM_MOVING          46
#define RAM_LOCK           47
#define RAM_PUNCH_L          48
#define RAM_PUNCH_H          49
#define RAM_CURRENT_L         68   //MX-64
#define RAM_CURRENT_H         69   //MX-64
#define RAM_TORQUE_CONTROL      70   //MX-64
#define RAM_GOAL_TORQUE_L       71   //MX-64
#define RAM_GOAL_TORQUE_H       72   //MX-64
#define RAM_GOAL_ACCEL        73   //MX-64

// Status Return Levels
   //////////////////////////////////////////////////////////
#define RETURN_NONE          0
#define RETURN_READ          1
#define RETURN_ALL          2

// Instruction Set
   //////////////////////////////////////////////////////////
#define DMXL_PING           1
#define DMXL_READ_DATA        2
#define DMXL_WRITE_DATA        3
#define DMXL_REG_WRITE  4
#define DMXL_ACTION  5
#define DMXL_RESET   6
#define DMXL_SYNC_WRITE  131   //0x83
#define DMXL_BULK_READ        146   //0x92

// Specials ////////////////////////////////////////////////////////
#define OFF  0
#define ON  1
#define DMXL_GOAL_SP_LENGTH   7
#define DMXL_ACTION_CHECKSUM     250
#define BROADCAST_ID  254
#define DMXL_START  255
#define Tx_MODE  1
#define Rx_MODE  0

//Length of commands
#define LENGTH_READ          4   //All read functions requiere only a length
     of 4 (Instruction + command adress + length of data + checksum)
#define LENGTH_ACTION         2   //Action only requires a length of 2 (
    Instruction + checksum)
#define LENGTH_PING          2   //Ping only requires a length of 2 (
    Instruction + checksum)
#define LENGTH_RESET         2   //Reset only requires a length of 2 (
    Instruction + checksum)
```

```
#define ONE_BYTE           1
#define TWO_BYTES           2

#include <inttypes.h>
#include <math.h>

#if defined(ARDUINO) && ARDUINO >= 100 // Arduino IDE Version
      #include "Arduino.h"
#else
      #include "WProgram.h"
#endif

class DynamixelClass {

private:

      // ----------------------------Platform-dependent pins
      #if (PLATFORM_ID==88) || defined(SPARK)
            uint8_t _directionPin = D15;    //For Duo or Photon
      #else
            uint8_t _directionPin = 4;     //For Leonardo
      #endif

      // ----------------------------Message Structure
      static const uint8_t MIN_RETURN_LEN = 5;  //Minimum length of return
         package {0xFF, 0xFF, ID, lengthMessage, Error_Byte, checksum}
      uint8_t Incoming_Byte, dataLSB, dataMSB, Checksum;
      int Error_Byte, data;

      // ----------------------------Buffers
      uint8_t PACKAGE[64] = {};
      uint8_t RESPONSE[64]= {};

      uint8_t statusReturnLevel = RETURN_ALL;   //Status return level. (
         default)2: Return for all commands. 1: Return only for the READ
         command. 0: No return (except PING)

      // ----------------------------Error definitions. Use negative values.
      static const int NO_ERROR = 0;
      static const int NO_SERVO_RESPONSE = -1;
      static const int LENGTH_INCORRECT = -2;

      // ----------------------------Variables regarding performance of
         DuoDMXL
      long _baudrateDMXL = 0;       //Current baudrate used with the servos
      static const uint16_t USBSERIAL_TIMEOUT = 2000; //Waiting time [ms]
         after Serial.begin() with PC. It allows the device to be recognized
          properly
```

```
    static const uint16_t DMXLSERIAL_TIMEOUT = 500; //Waiting time [ms]
        after Serial1.begin() with servos
    uint8_t TIME_OUT = 50;      //Waiting time (milliseconds) for the
        incomming data from servomotor. Recommended value:50
    uint16_t COOL_DOWN = 0;       //Cool down period (milliseconds) before
         sending another command to the dynamixel servomotor

    // ----------------------------flags
    bool _response_within_timeout = true;   //Assume every byte of the
        response is within time

    // ----------------------------Communication related functions
    int readInformation(void);

    bool waitData(int, int);
    bool waitData(int);
    bool waitData();

public:

    // ----------------------------General functions

    int sendWord(uint8_t ID, uint8_t address, int params, int noParams,
        uint8_t instruction);
    int sendWords(uint8_t IDs[], uint8_t noIDs, uint8_t address, int
        params[], int noParams);
    int readWord(uint8_t ID, uint8_t address, int noParams);
    void readWords(uint8_t IDs[], uint8_t noIDs, uint8_t address, int
        noParams, int *response);

    void begin(long baud, uint8_t directionPin);
    void begin(long baud);
    void end(void);

    int reset(uint8_t ID);
    int ping(uint8_t ID);
    void action(uint8_t ID);

    // ----------------------------EEPROM Area Instructions

    int readModel(uint8_t ID);
    int readFirmware(uint8_t ID);
    int setID(uint8_t ID, uint8_t newID);
    int readID(uint8_t ID);
    int setBD(uint8_t ID, long baud);
    int setBDTable(uint8_t ID, uint8_t baud);
    int readBD(uint8_t ID);
    int setRDT(uint8_t ID, uint8_t RDT);
```

```
int readRDT(uint8_t ID);
int setCWAngleLimit(uint8_t ID, int limit);
int readCWAngleLimit(uint8_t ID);
int setCCWAngleLimit(uint8_t ID, int limit);
int readCCWAngleLimit(uint8_t ID);
int setTempLimit(uint8_t ID, uint8_t Temperature);
int readTempLimit(uint8_t ID);
int setLowVoltageLimit(uint8_t ID, uint8_t lowVoltage);
int readLowVoltageLimit(uint8_t ID);
int setHighVoltageLimit(uint8_t ID, uint8_t highVoltage);
int readHighVoltageLimit(uint8_t ID);
int setMaxTorque(uint8_t ID, int MaxTorque);
int readMaxTorque(uint8_t ID);
int setSRL(uint8_t ID, uint8_t SRL);
int readSRL(uint8_t ID);
int setBoardSRL(uint8_t SRL);
int setAlarmLED(uint8_t ID, uint8_t alarm);
int readAlarmLED(uint8_t ID);
int setShutdownAlarm(uint8_t ID, uint8_t SALARM);
int readShutdownAlarm(uint8_t ID);
int setMultiTurnOffset(uint8_t ID, int offset);
int readMultiTurnOffset(uint8_t ID);
int setResolutionDivider(uint8_t ID, uint8_t divider);
int readResolutionDivider(uint8_t ID);

// ---------------------------RAM Area Instructions

int torqueEnable(uint8_t ID, bool Status);
int torqueEnableStatus(uint8_t ID);
int ledStatus(uint8_t ID, bool Status);
int setGainD(uint8_t ID, int gain);
int readGainD(uint8_t ID);
int setGainI(uint8_t ID, int gain);
int readGainI(uint8_t ID);
int setGainP(uint8_t ID, int gain);
int readGainP(uint8_t ID);
int move(uint8_t ID, int Position);
int move(uint8_t IDs[], uint8_t noIDs, int Positions[]);
//int moveSpeed(uint8_t ID, int Position, int Speed);
int setMovingSpeed(uint8_t ID, int speed);
int readMovingSpeed(uint8_t ID);
int setTorqueLimit(uint8_t ID, int torque);
int readTorqueLimit(uint8_t ID);
int readPosition(uint8_t ID);
void readPosition(uint8_t IDs[], uint8_t noIDs, int *positions);
int readSpeed(uint8_t ID);
int readLoad(uint8_t ID);
int readVoltage(uint8_t ID);
```

```
    int readTemperature(uint8_t ID);
    int registeredStatus(uint8_t ID);
    int moving(uint8_t ID);
    int lockEEPROM(uint8_t ID);
    int setPunch(uint8_t ID, int Punch);
    int readPunch(uint8_t ID);
    int readCurrent(uint8_t ID);
    int torqueControl( uint8_t ID, bool enable);
    int readTorqueControl( uint8_t ID);
    int setGoalTorque(uint8_t ID, int torque);
    int setGoalAccel(uint8_t ID, uint8_t accel);

    // ---------------------------Custom functions

    int setAng(uint8_t ID, float angle);
    int setAng(uint8_t ID, float angle, char unit);
    void setDirectionPin(uint8_t pin);
    uint8_t getDirectionPin();
    void setBaudrateDMXL(long baud);
    long getBaudrateDMXL();
    void configureServo(uint8_t ID, uint8_t newID, long baud);
    void setAngleLimit(uint8_t ID, int CWLimit, int CCWLimit);
    void setWheelMode(uint8_t ID, bool enable);
    void setJointMode(uint8_t ID);
    void setDIP(uint8_t ID, int gainD, int gainI, int gainP);
    int findByBaudRate(long baudRate);
    int findByID(uint8_t id, uint8_t directionPin);
    void findServo(uint8_t directionPin);
    void changeTimeOut(uint8_t newTimeOut);
    void changeCoolDown(uint16_t newCoolDown);
    void servoIntroduction(uint8_t ID);

    // ---------------------------Multi-compatibility functions

    void sendData(uint8_t);
    void sendDataBuff(uint8_t*, uint8_t);
    int availableData(void);
    uint8_t readData(void);
    uint8_t peekData(void);
    void beginCom(long);
    void endCom(void);
    void serialFlush(void);
    void delayms(unsigned int);
    void delayus(unsigned int);
};

extern DynamixelClass Dynamixel;
```

```
#endif //DuoDMXL_h
```

Listing E.3: DuoDMXL.cpp

```
/*
DuoDMXL v2.0
MX-64AR Half Duplex USART/RS-485 Communication Library
-------------------------------------------------------------------------------

Target Boards:
      Redbear Duo
      Particle Photon (not tested)
      Arduino Leonardo
      or any other board with two hardware serial ports (soft serial not
         tested)

DuoDMXL.h:
      Definitions of constants and methods for class DynamixelClass

      Initially based on Savage's DynamixelSerial Library
      http://savageelectronics.blogspot.jp/2011/01/arduino-y-dynamixel-ax
         -12.html
-------------------------------------------------------------------------------


Copyright(c) 2016 Fabian Eugenio Reyes Pinner
Created by Fabian E. Reyes Pinner on 2016-06-01 (August 1st, 2016)

This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU Lesser General Public License as published
      by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
   GNU Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public License
   along with this program. If not, see <http://www.gnu.org/licenses/>.

-------------------------------------------------------------------------------

 */

#include "DuoDMXL.h"
```

```
// Macro for Comunication Flow Control
#if (PLATFORM_ID==88) || defined(SPARK)
      #define setDPin(DirPin,Mode) (pinMode(DirPin,Mode))  // Select the
         Switch to TX/RX Mode Pin
      #define switchCom(DirPin,Mode)  (digitalWrite(DirPin,Mode)) // Switch
         to TX/RX Mode
      #define printPC(value)    (Serial.print(value))   // Print to the PC
      #define printlnPC(value)   (Serial.println(value))  // Print to the PC
#else
      #define setDPin(DirPin,Mode) (pinMode(DirPin,Mode)) // Select the
         Switch to TX/RX Mode Pin
      #define switchCom(DirPin,Mode) (digitalWrite(DirPin,Mode)) // Switch
         to TX/RX Mode
      #define printPC(value)    (Serial.print(value))   // Print to the PC
      #define printlnPC(value)   (Serial.println(value))  // Print to the PC
#endif

// Private Methods
   //////////////////////////////////////////////////////////////

//General function to read the status package from the servo
int DynamixelClass::readInformation(void)
{
      unsigned long startTime = millis();
      int processTime, lengthMessage, dataLength;

      //Reset flag
      _response_within_timeout = true;

      //Reset the buffer
      memset(RESPONSE, 0, 64);

      //DuoDMXL v.1.6. Do nothing until we have the start bytes, ID, length
         of the message (four bytes). Even at 9600bps, it should take around
          3.33[ms] for the four bytes to arrive
      waitData(4, (int) TIME_OUT);

      //DuoDMXL v.1.0+. Only proceed if there was no problem in the
         communication
      while ( (availableData() > 0) && _response_within_timeout){
            Incoming_Byte = readData();          //First byte of the header

            //For now, DuoDMXL assumes there are no problems in the rest of
               the communication
            if ( (Incoming_Byte == 255) && (peekData() == 255) ){

                  //Fill buffer for debugging
                  RESPONSE[0] = Incoming_Byte;
```

```
                RESPONSE[1] = readData();  //Second byte of the header
                RESPONSE[2] = readData();  //Dynamixel ID
                RESPONSE[3] = lengthMessage = readData();     //Length of
                    the message
                dataLength = lengthMessage-2;

                waitData((int) TIME_OUT);         //Do nothing until the
                    next byte is in the buffer
                RESPONSE[4] = Error_Byte = readData();  //Error

                if(dataLength == 0){
                      data = Error_Byte;          //No data is returned.
                          Send Error_Byte. This is the common response when
                           sending commands
                }
                else if( dataLength == 1 ){
                      waitData((int) TIME_OUT);

                      dataLSB = readData();       //LSB of the data
                      data = (int) dataLSB;

                      RESPONSE[5] = dataLSB;
                }
                else if(dataLength == 2){
                      waitData(1, TIME_OUT);

                      dataLSB = readData();       //LSB of the data
                      dataMSB = readData();         //MSB of the data
                      data = dataMSB << 8;
                      data = data + dataLSB;

                      RESPONSE[5] = dataLSB;
                      RESPONSE[6] = dataMSB;
                }
                else{
                      data = LENGTH_INCORRECT;        //The length was not
                           correct or there was some problem
                }

                waitData((int) TIME_OUT);
                RESPONSE[MIN_RETURN_LEN + dataLength] = readData();   //
                    checksum

                delayms(COOL_DOWN);
                return (data);
          }
     }
```

```
      //If there was a TIME_OUT and not enough bytes, there was an error in
         communication
      delayms(COOL_DOWN);
      return (NO_SERVO_RESPONSE);           // No servo Response
}

//Wait for a certain number of bytes 'length', for a certain amount of time
   'timeLimit' [ms]
bool DynamixelClass::waitData(int length, int timeLimit){

      unsigned long startTime = millis();
      int processTime;
   bool response_within_timeout = true;

   while( (availableData() <=length) && response_within_timeout){
      processTime = (int) millis();
      processTime = processTime - startTime; //time since this function
         started

      response_within_timeout = (bool) (processTime <= timeLimit ); //is the
          communication within the allowed time?
   }

      //Set global flag
      _response_within_timeout = response_within_timeout;
   return response_within_timeout;
}

//Wait for one byte, for a certain amount of time 'timeLimit' [ms]
bool DynamixelClass::waitData(int timeLimit){
   return waitData(0, timeLimit);
}

//keep waiting until there is at least one byte in the buffer
bool DynamixelClass::waitData(){
   while( !availableData() ){}
      return true;
}

//-----------Public Methods-----------------------------------------------------

/*
Function to set (write) the value of a servo's address.
noParams should be ONE_BYTE or TWO_BYTES, depending on how many bytes we
   need to send
In general, instruction should be either DMXL_WRITE_DATA or DMXL_REG_WRITE
*/
```

```
int DynamixelClass::sendWord(uint8_t ID, uint8_t address, int param, int
   noParams, uint8_t instruction){
      uint8_t param_MSB, param_LSB, length, lengthPackage;
      uint8_t *package = NULL;

      param_MSB = param >> 8;
      param_LSB = param;

      //Based on numbers of parameters, decide size of outgoing package and
         calculate checksum
      if(noParams == 0){
            length = 2;          //instruction + checksum = 2
            Checksum = (~(ID + length + instruction))&0xFF;

            lengthPackage = 6;
            package = new uint8_t[lengthPackage];

            package[5] = Checksum;
      }
      else if(noParams == ONE_BYTE){
            length = 4;          //instruction + address + param_LSB +
               checksum = noParams + 3 = 4
            Checksum = (~(ID + length + instruction + address + param_LSB))
               &0xFF;

            lengthPackage = 8;
            package = new uint8_t[lengthPackage];

            package[5] = address;
            package[6] = param_LSB;
            package[7] = Checksum;
      }
      else if(noParams == TWO_BYTES){
            length = 5;          //instruction + address + param_LSB +
               param_MSB + checksum = noParams + 3 = 5
            Checksum = (~(ID + length + instruction + address + param_LSB +
               param_MSB))&0xFF;

            lengthPackage = 9;
            package = new uint8_t[lengthPackage];

            package[5] = address;
            package[6] = param_LSB;
            package[7] = param_MSB;
            package[8] = Checksum;
      }
```

```
      //Rest of the package, common for either a 0, ONE_BYTE or TWO_BYTES
         package
      package[0] = DMXL_START;
      package[1] = DMXL_START;
      package[2] = ID;
      package[3] = length;
      package[4] = instruction;

      switchCom(_directionPin,Tx_MODE);
      sendDataBuff(package, lengthPackage);
      serialFlush();
      switchCom(_directionPin,Rx_MODE);

      //free(package);
      delete[] package;

      if( (instruction == DMXL_PING) || ((statusReturnLevel==RETURN_ALL) &&
         (ID!=BROADCAST_ID)) ){
            return(readInformation());
      }
      else{
            return(NO_ERROR);
      }
}

//Function to set the value of a several servos' address. noIDs is how many
   servos we are communicating with
//params is an array with values (one for each servo). noParams should be
   ONE_BYTE or TWO_BYTES, depending on how many bytes we need to send per
   servo
int DynamixelClass::sendWords(uint8_t IDs[], uint8_t noIDs, uint8_t address,
    int params[], int noParams){

      //1- Prepare information
      uint8_t length = (noParams+1)*noIDs + 4;    //instruction + address +
         noParams + (1 + noParams)*noIDs + checksum = (1 + noParams)*noIDs +
          4
      uint8_t lengthPackage = length + 4;      //DMXL_START + DMXL_START +
         BROADCAST_ID + length + {package}
      uint8_t package[lengthPackage] = {};
      uint16_t tempChecksum = 0;

      //2- Buffer to hold all the information
      package[0]=DMXL_START;
      package[1]=DMXL_START;
      package[2]=BROADCAST_ID;
      package[3]=length;
      package[4]=DMXL_SYNC_WRITE;
```

```
      package[5]=address;
      package[6]=noParams;

      for(uint8_t i=0; i<noIDs; i++){
            //Save ID
            package[7 + i*(1 + noParams)] = IDs[i];

            //Save LSB and MSB
            for(uint8_t j=0; j<noParams; j++){
                  package[7 + i*(1 + noParams) + 1 + j] = (uint8_t) ( params
                      [i] >> (8*j) );
            }
      }

      //Obtain cheksum and save it in the last position of the buffer.
         Checksum starts at the ID
      for(uint8_t i=2; i<(lengthPackage-1); i++){
            tempChecksum += package[i];
      }
      Checksum = (~tempChecksum)&0xFF;
      package[lengthPackage-1] = Checksum;

      //Send the package
      switchCom(_directionPin,Tx_MODE);
      sendDataBuff(package, lengthPackage);
      serialFlush();
      switchCom(_directionPin,Rx_MODE);
}

//Function to read the value of a servo's address. noParams should be
   ONE_BYTE or TWO_BYTES, depending on how many bytes we need
int DynamixelClass::readWord(uint8_t ID, uint8_t address, int noParams){

      Checksum = (~(ID + LENGTH_READ + DMXL_READ_DATA + address + noParams))
         &0xFF;

      //Prepare a buffer with all information
      uint8_t package[8] = {DMXL_START, DMXL_START, ID, LENGTH_READ,
         DMXL_READ_DATA, address, noParams, Checksum};

      switchCom(_directionPin,Tx_MODE);
      sendDataBuff(package, 8);
      serialFlush();
      switchCom(_directionPin,Rx_MODE);

      //Depending on the current value of statusReturnLevel read or skip the
          status package
```

```
      //if reading a word, then we only expect a package when
         statusReturnLevel is RETURN_ALL or RETURN_READ
      if(statusReturnLevel==RETURN_NONE){
            return(NO_ERROR);
      }
      else{
            return(readInformation());
      }
}

//Function to read the value of a several servos. noParams should be
   ONE_BYTE or TWO_BYTES, depending on how many bytes we need
//This function assumes address and noParams is the same for all servos. It
   is necessary to pass a previously allocated array
void DynamixelClass::readWords(uint8_t IDs[], uint8_t noIDs, uint8_t address
   , int noParams, int *response){

      //1- Prepare information
      uint8_t length = 3*noIDs + 3;     //instruction + 0x00 + noParams1 +
         ID1 + address1 + ... + noParamsN + IDN + addressN + Checksum
      uint8_t lengthPackage = length + 4;    //DMXL_START + DMXL_START +
         BROADCAST_ID + length + ...
      uint8_t package[lengthPackage] = {};
      uint16_t tempChecksum = 0;

      //2- I need to create a buffer to hold all the information
      package[0] = DMXL_START;
      package[1] = DMXL_START;
      package[2] = BROADCAST_ID;
      package[3] = length;
      package[4] = DMXL_BULK_READ;
      package[5] = 0x00;

      for(uint8_t i=0; i<noIDs; i++){
            package[6 + i*3] = noParams;
            package[6 + i*3 + 1] = IDs[i];
            package[6 + i*3 + 2] = address;
      }

      //3- Obtain cheksum and save it in the last position of the buffer.
         Checksum starts at the ID
      for(uint8_t i=2; i<(lengthPackage-1); i++){
            tempChecksum += package[i];
      }

      Checksum = (~tempChecksum)&0xFF;
      package[lengthPackage-1] = Checksum;
```

```
      //4- Send the package
      switchCom(_directionPin,Tx_MODE);
      sendDataBuff(package, lengthPackage);
      serialFlush();
      switchCom(_directionPin,Rx_MODE);

      //5- Read the return package and save it
      for(uint8_t i=0; i<noIDs; i++){
            response[i] = readInformation();
      }

}

//Initialize communication with the servos with a user-defined pin for the
   data direction control
//By default, always set status return level as RETURN_ALL. TODO: Read SRL
   from either the servos, or the EEPROM
void DynamixelClass::begin(long baud, uint8_t directionPin){

      //Save private variables
      setBaudrateDMXL(baud);
      setDirectionPin(directionPin);

      //Start connection using the selected pin
      setDPin(_directionPin, OUTPUT);
      beginCom(baud);
      delayms(DMXLSERIAL_TIMEOUT);

      //Set the SRL to RETURN_ALL
      setSRL(BROADCAST_ID, RETURN_ALL);
}

//Initialize communication with the servos with a pre-defined pin (D15) for
   the data direction control
void DynamixelClass::begin(long baud){

      //Save private variables
      setBaudrateDMXL(baud);

      //Start communication with the default pin
      setDPin(_directionPin, OUTPUT);
      beginCom(baud);
      delayms(DMXLSERIAL_TIMEOUT);

      //Set the SRL to RETURN_ALL
      setSRL(BROADCAST_ID, RETURN_ALL);
}
```

```
//End communication
void DynamixelClass::end(){
      endCom();
}

//Function inherited from Savage's library
int DynamixelClass::reset(uint8_t ID){
      //Upon reset, SRL should be set to RETURN_ALL
      statusReturnLevel = RETURN_ALL;
      return(sendWord(ID, 0, 0, 0, DMXL_RESET));
}

//Ping a servo
int DynamixelClass::ping(uint8_t ID){
      return(sendWord(ID, 0, 0, 0, DMXL_PING));
}

//Send the ACTION Instruction
void DynamixelClass::action(uint8_t ID){
      sendWord(ID, 0, 0, 0, DMXL_ACTION);
}

//Function to read the servo model. EEPROM Address 0(x00) and 1(0x01)
int DynamixelClass::readModel(uint8_t ID){
      return(readWord(ID, EEPROM_MODEL_NUMBER_L, TWO_BYTES));
}

//Function to read the version of the firmware. EEPROM Address 2(0x02)
int DynamixelClass::readFirmware(uint8_t ID){
      return(readWord(ID, EEPROM_VERSION, ONE_BYTE));
}

//Function to set the ID of the servo. EEPROM Address 3(0x03)
int DynamixelClass::setID(uint8_t ID, uint8_t newID){
      return(sendWord(ID, EEPROM_ID, newID, ONE_BYTE, DMXL_WRITE_DATA));
}

//Function to read the ID of the servo. EEPROM Address 3(0x03)
int DynamixelClass::readID(uint8_t ID){
      return(readWord(ID, EEPROM_ID, ONE_BYTE));
}

//Function to set baudrate. EEPROM Address 4(0x04).
//The baudrate change takes effect immediately. You need to use end() and
   then begin() with the new baudrate
int DynamixelClass::setBD(uint8_t ID, long baud){
      uint8_t Baud_Rate = round( (2000000.0/((float) baud)) -1 );
```

```
      return(sendWord(ID, EEPROM_BAUD_RATE, Baud_Rate, ONE_BYTE,
         DMXL_WRITE_DATA));
}

//Function to set baudrate based on the manual's table. EEPROM Address 4(0
   x04)
//The baudrate change takes effect immediately. You need to use end() and
   begin() with the new baudrate
int DynamixelClass::setBDTable(uint8_t ID, uint8_t baud){
      return(sendWord(ID, EEPROM_BAUD_RATE, baud, ONE_BYTE, DMXL_WRITE_DATA)
         );
}

//Function to read the setting of the baudrate. EEPROM Address 4(0x04)
int DynamixelClass::readBD(uint8_t ID){
      return(readWord(ID, EEPROM_BAUD_RATE, ONE_BYTE));
}

//Set the Return Delay Time (RDT) in microseconds. EEPROM Address 5(0x05)
int DynamixelClass::setRDT(uint8_t ID, uint8_t RDT){
      return(sendWord(ID, EEPROM_RETURN_DELAY_TIME, RDT/2, ONE_BYTE,
         DMXL_WRITE_DATA));
}

//Read the Return Delay Time (RDT) value. EEPROM Address 5(0x05)
int DynamixelClass::readRDT(uint8_t ID){
      return(readWord(ID, EEPROM_RETURN_DELAY_TIME, ONE_BYTE));
}

//Set the value for the CW Angle limit. EEPROM Address 6(0x06) and 7(0x07)
int DynamixelClass::setCWAngleLimit(uint8_t ID, int limit){
      return(sendWord(ID, EEPROM_CW_ANGLE_LIMIT_L, limit, TWO_BYTES,
         DMXL_WRITE_DATA));
}

//Read the value for the CW Angle limit. EEPROM Address 6(0x06) and 7(0x07)
int DynamixelClass::readCWAngleLimit(uint8_t ID){
      return(readWord(ID, EEPROM_CW_ANGLE_LIMIT_L, TWO_BYTES));
}

//Set the value for the CCW Angle limit. EEPROM Address 8(0x08) and 9(0x09)
int DynamixelClass::setCCWAngleLimit(uint8_t ID, int limit){
      return(sendWord(ID, EEPROM_CCW_ANGLE_LIMIT_L, limit, TWO_BYTES,
         DMXL_WRITE_DATA));
}

//Read the value for the CCW Angle limit. EEPROM Address 8(0x08) and 9(0x09)
int DynamixelClass::readCCWAngleLimit(uint8_t ID){
```

```
      return(readWord(ID, EEPROM_CCW_ANGLE_LIMIT_L, TWO_BYTES));
}

//Set the limit temperature. EEPROM Address 11(0x0B)
int DynamixelClass::setTempLimit(uint8_t ID, uint8_t Temperature){
      return(sendWord(ID, EEPROM_LIMIT_TEMPERATURE, Temperature, ONE_BYTE,
         DMXL_WRITE_DATA));
}

//Read the limit temperature. EEPROM Address 11(0x0B)
int DynamixelClass::readTempLimit(uint8_t ID){
      return(readWord(ID, EEPROM_LIMIT_TEMPERATURE, ONE_BYTE));
}

//Set the lowest voltage limit. EEPROM Address 12(0x0C)
int DynamixelClass::setLowVoltageLimit(uint8_t ID, uint8_t lowVoltage){
      return(sendWord(ID, EEPROM_DOWN_LIMIT_VOLTAGE, lowVoltage, ONE_BYTE,
         DMXL_WRITE_DATA));
}

//Read the lowest voltage limit. EEPROM Address 12(0x0C)
int DynamixelClass::readLowVoltageLimit(uint8_t ID){
      return(readWord(ID, EEPROM_DOWN_LIMIT_VOLTAGE, ONE_BYTE));
}

//Set the highest voltage limit. EEPROM Address 13(0x0D)
int DynamixelClass::setHighVoltageLimit(uint8_t ID, uint8_t highVoltage){
      return(sendWord(ID, EEPROM_UP_LIMIT_VOLTAGE, highVoltage, ONE_BYTE,
         DMXL_WRITE_DATA));
}

//Read the highest voltage limit. EEPROM Address 13(0x0D)
int DynamixelClass::readHighVoltageLimit(uint8_t ID){
      return(readWord(ID, EEPROM_UP_LIMIT_VOLTAGE, ONE_BYTE));
}

//Set the maximum torque. EEPROM Address 14(0x0E) and 15(0x0F)
int DynamixelClass::setMaxTorque(uint8_t ID, int MaxTorque){
      return(sendWord(ID, EEPROM_MAX_TORQUE_L, MaxTorque, TWO_BYTES,
         DMXL_WRITE_DATA));
}

//Read the maximum torque. EEPROM Address 14(0x0E) and 15(0x0F)
int DynamixelClass::readMaxTorque(uint8_t ID){
      return(readWord(ID, EEPROM_MAX_TORQUE_L, TWO_BYTES));
}

//Set the Status Return Level. EEPROM Address 16(0x10).
```

```
//DuoDMXL assumes that upon reset, all servos have RETURN_ALL. If the value
   was changed in a previos session, use setSRL(BROADCAST_ID, 2) or
   setBoardSRL(SRL)
//The change in SRL takes place beginning with the NEXT communication. Even
   if sending RETURN_NONE, you may still get a status return
int DynamixelClass::setSRL(uint8_t ID, uint8_t SRL){

      //Send the new desired status return level
      int error = sendWord(ID, EEPROM_RETURN_LEVEL, SRL, ONE_BYTE,
         DMXL_WRITE_DATA);
      statusReturnLevel = SRL;

      return(error);
}

//Read the Status Return Level value. EEPROM Address 16(0x10)
int DynamixelClass::readSRL(uint8_t ID){
      return(readWord(ID, EEPROM_RETURN_LEVEL, ONE_BYTE));
}

//Forcefully set SRL value saved in the DUO, instead of the servo's SRL
   value
int DynamixelClass::setBoardSRL(uint8_t SRL){
      //change the current value of statusReturnLevel
      statusReturnLevel = SRL;
      return(NO_ERROR);
}

//Set Alarm LED. EEPROM Address 17(0x11)
int DynamixelClass::setAlarmLED(uint8_t ID, uint8_t alarm){
      return(sendWord(ID, EEPROM_ALARM_LED, alarm, ONE_BYTE, DMXL_WRITE_DATA
         ));
}

//Read Alarm LED value. EEPROM Address 17(0x11)
int DynamixelClass::readAlarmLED(uint8_t ID){
      return(readWord(ID, EEPROM_ALARM_LED, ONE_BYTE));
}

//Set Shutdown alarm. EEPROM Address 18(0x12)
int DynamixelClass::setShutdownAlarm(uint8_t ID, uint8_t SALARM){
      return(sendWord(ID, EEPROM_ALARM_SHUTDOWN, SALARM, ONE_BYTE,
         DMXL_WRITE_DATA));
}

//Read Shutdown alarm value. EEPROM Address 18(0x12)
int DynamixelClass::readShutdownAlarm(uint8_t ID){
      return(readWord(ID, EEPROM_ALARM_SHUTDOWN, ONE_BYTE));
```

```
}

//Set the multi-turn offset values. EEPROM ADDRESS: 20(0x14) and 21(0x15)
int DynamixelClass::setMultiTurnOffset(uint8_t ID, int offset){
      return(sendWord(ID, EEPROM_TURN_OFFSET_L, offset, TWO_BYTES,
         DMXL_WRITE_DATA));
}

//Read the multi-turn offset values. EEPROM ADDRESS: 20(0x14) and 21(0x15)
int DynamixelClass::readMultiTurnOffset(uint8_t ID){
      return(readWord(ID, EEPROM_TURN_OFFSET_L, TWO_BYTES));
}

//Set the resolution divider value. EEPROM ADDRESS: 22(0x16)
int DynamixelClass::setResolutionDivider(uint8_t ID, uint8_t divider){
      return(sendWord(ID, EEPROM_RESOLUTION_DIV, divider, ONE_BYTE,
         DMXL_WRITE_DATA));
}

//Read the resolution divider value. EEPROM ADDRESS: 22(0x16)
int DynamixelClass::readResolutionDivider(uint8_t ID){
      return(readWord(ID, EEPROM_RESOLUTION_DIV, ONE_BYTE));
}

//FUNCTIONS TO ACCESS COMMANDS IN THE RAM AREA

//Function to turn ON or OFF torque. RAM Address 24(0x18)
int DynamixelClass::torqueEnable( uint8_t ID, bool Status){
      return(sendWord(ID, RAM_TORQUE_ENABLE, (int) Status, ONE_BYTE,
         DMXL_WRITE_DATA));
}

//Function to check if the servo generates torque. RAM Address 24(0x18)
int DynamixelClass::torqueEnableStatus( uint8_t ID){
      return(readWord(ID, RAM_TORQUE_ENABLE, ONE_BYTE));
}

//Function to turn ON or OFF the servo's LED. RAM Address 25(0x19)
int DynamixelClass::ledStatus(uint8_t ID, bool Status){
      return(sendWord(ID, RAM_LED, (int) Status, ONE_BYTE, DMXL_WRITE_DATA))
         ;
}

//Function to set the value of the Derivative gain. RAM Address 26(0x1A)
int DynamixelClass::setGainD(uint8_t ID, int gain){
      return(sendWord(ID, RAM_D_GAIN, gain, ONE_BYTE, DMXL_WRITE_DATA));
}
```

```
//Function to read the value of the Derivative gain. RAM Address 26(0x1A)
int DynamixelClass::readGainD(uint8_t ID){
      return(readWord(ID, RAM_D_GAIN, ONE_BYTE));
}

//Function to set the value of the Integral gain. RAM Address 27(0x1B)
int DynamixelClass::setGainI(uint8_t ID, int gain){
      return(sendWord(ID, RAM_I_GAIN, gain, ONE_BYTE, DMXL_WRITE_DATA));
}

//Function to read the value of the Integral gain. RAM Address 27(0x1B)
int DynamixelClass::readGainI(uint8_t ID){
      return(readWord(ID, RAM_I_GAIN, ONE_BYTE));
}

//Function to set the value of the Proportional gain. RAM Address 28(0x1C)
int DynamixelClass::setGainP(uint8_t ID, int gain){
      return(sendWord(ID, RAM_P_GAIN, gain, ONE_BYTE, DMXL_WRITE_DATA));
}

//Function to read the value of the Proportional gain. RAM Address 28(0x1C)
int DynamixelClass::readGainP(uint8_t ID){
      return(readWord(ID, RAM_P_GAIN, ONE_BYTE));
}

//Function to move servo to a specific position. RAM Address 30(0x1E) and
   31(0x1F)
int DynamixelClass::move(uint8_t ID, int Position){
      return(sendWord(ID, RAM_GOAL_POSITION_L, Position, TWO_BYTES,
         DMXL_WRITE_DATA));
}

//Function to move servos to a specific positions. RAM Address 30(0x1E) and
   31(0x1F)
int DynamixelClass::move(uint8_t IDs[], uint8_t noIDs, int Positions[]){
      return(sendWords(IDs, noIDs, RAM_GOAL_POSITION_L, Positions, TWO_BYTES
         ));
}

//Function to set the desired moving speed. RAM Address 32(0x20) and 33(0x21
   )
int DynamixelClass::setMovingSpeed(uint8_t ID, int speed){
      return(sendWord(ID, RAM_GOAL_SPEED_L, speed, TWO_BYTES,
         DMXL_WRITE_DATA));
}

//Function to read the desired moving speed. RAM Address 32(0x20) and 33(0
   x21)
```

```
int DynamixelClass::readMovingSpeed(uint8_t ID){
      return(readWord(ID, RAM_GOAL_SPEED_L, TWO_BYTES));
}

//Function to set the value of the goal torque. RAM Address 34(0x22) and
   35(0x23)
int DynamixelClass::setTorqueLimit(uint8_t ID, int torque){
      return(sendWord(ID, RAM_TORQUE_LIMIT_L, torque, TWO_BYTES,
         DMXL_WRITE_DATA));
}

//Function to read the value of the goal torque. RAM Address 34(0x22) and
   35(0x23)
int DynamixelClass::readTorqueLimit(uint8_t ID){
      return(readWord(ID, RAM_TORQUE_LIMIT_L, TWO_BYTES));
}

//Read the actual position of one servo. RAM Address 36(0x24) and 37(0x25)
int DynamixelClass::readPosition(uint8_t ID){
      return(readWord(ID, RAM_PRESENT_POSITION_L, TWO_BYTES));
}

//Read the actual position of several servos. RAM Address 36(0x24) and 37(0
   x25)
void DynamixelClass::readPosition(uint8_t IDs[], uint8_t noIDs, int *
   positions){
      readWords(IDs, noIDs, RAM_PRESENT_POSITION_L, TWO_BYTES, positions);
}

//Read the actual speed. RAM Address 38(0x26) and 39(0x27)
int DynamixelClass::readSpeed(uint8_t ID){
      return(readWord(ID, RAM_PRESENT_SPEED_L, TWO_BYTES));
}

//Read the load. RAM Address 40(0x28) and 41(0x29)
int DynamixelClass::readLoad(uint8_t ID){
      return(readWord(ID, RAM_PRESENT_LOAD_L, TWO_BYTES));
}

//Function to read the voltage. RAM Address 42(0x2A)
int DynamixelClass::readVoltage(uint8_t ID){
      return(readWord(ID, RAM_PRESENT_VOLTAGE, ONE_BYTE));
}

//Function to read the Temperature. RAM Address 43(0x2B)
int DynamixelClass::readTemperature(uint8_t ID){
      return(readWord(ID, RAM_PRESENT_TEMPERATURE, ONE_BYTE));
}
```

```
//Check if there is an instruction registered. RAM Address 44(0x2C)
int DynamixelClass::registeredStatus(uint8_t ID){
      return(readWord(ID, RAM_REGISTERED_INSTRUCTION, ONE_BYTE));
}

//Check if goal position command is being executed (Address 0x30?). RAM
   Address 46(0x2E)
int DynamixelClass::moving(uint8_t ID){
      return(readWord(ID, RAM_MOVING, ONE_BYTE));
}

//Locks the EEPROM. RAM Address 47(0x2F)
int DynamixelClass::lockEEPROM(uint8_t ID){
      return(sendWord(ID, RAM_LOCK, 1, ONE_BYTE, DMXL_WRITE_DATA));
}

//RAM Address 48(0x30) and 49(0x31)
int DynamixelClass::setPunch(uint8_t ID, int Punch){
      return(sendWord(ID, RAM_PUNCH_L, Punch, TWO_BYTES, DMXL_WRITE_DATA));
}

//RAM Address 48(0x30) and 49(0x31)
int DynamixelClass::readPunch(uint8_t ID){
      return(readWord(ID, RAM_PUNCH_L, TWO_BYTES));
}

//Function to read the current. RAM ADDRESS: 68(0x44) and 69(0x45)
int DynamixelClass::readCurrent(uint8_t ID){
      return(readWord(ID, RAM_CURRENT_L, TWO_BYTES));
}

//Torque control mode enable. RAM ADDRESS: 70(0x46)
int DynamixelClass::torqueControl( uint8_t ID, bool enable){
      return(sendWord(ID, RAM_TORQUE_CONTROL, (int) enable, ONE_BYTE,
         DMXL_WRITE_DATA));
}

//Read the Torque control mode status. RAM ADDRESS: 70(0x46)
int DynamixelClass::readTorqueControl( uint8_t ID){
      return(readWord(ID, RAM_TORQUE_CONTROL, ONE_BYTE));
}

//Function to set the goal torque. RAM ADDRESS: 71(0x47) and 72(0x48)
int DynamixelClass::setGoalTorque(uint8_t ID, int torque){
      return(sendWord(ID, RAM_GOAL_TORQUE_L, torque, TWO_BYTES,
         DMXL_WRITE_DATA));
}
```

```
//Function to set goal acceleration/ RAM ADDRESS: 73(0x49)
int DynamixelClass::setGoalAccel(uint8_t ID, uint8_t accel){
      return(sendWord(ID, RAM_GOAL_ACCEL, accel, ONE_BYTE, DMXL_WRITE_DATA))
         ;
}

//CUSTOM FUNCTIONS-------------------------------------------------------

//Function to move servo to a specific angle [deg]. RAM Address 30(0x1E) and
    31(0x1F)
int DynamixelClass::setAng(uint8_t ID, float angle){
      //transform angle into 0-4095 values
      int Position;
      Position = (int) ( map(angle, 0.0, 360.0, 0.0, 4095.0) );
      Position = constrain(Position, 0, 4095);

      return(sendWord(ID, RAM_GOAL_POSITION_L, Position, TWO_BYTES,
         DMXL_WRITE_DATA));
}

//Function to move servo to a specific angle [deg]. RAM Address 30(0x1E) and
    31(0x1F)
//unit specify the input units: 'b': bit-value, 'd': degrees, 'r': radians
int DynamixelClass::setAng(uint8_t ID, float angle, char unit){

      int Position, error;

      if(unit == 'b'){
            Position = (int) angle;
            error = move(ID, Position);
      }
      else if(unit == 'd'){
            error = setAng(ID, angle);
      }
      else if(unit == 'r'){
            Position = (int) ( map(angle, 0.0, 2*M_PI, 0.0, 4095.0) );
            Position = constrain(Position, 0, 4095);
            error = move(ID, Position);
      }
      else{
            error = -2;
      }

      return(error);
}

//Set the direction pin of the serial communication with DMXL
```

```
void DynamixelClass::setDirectionPin(uint8_t pin){
      _directionPin = pin;
}

//Get the direction pin of the serial communication with DMXL
uint8_t DynamixelClass::getDirectionPin(){
      return(_directionPin);
}

//Set the baudrate of the serial communication with DMXL without ending/
   beginning serial
void DynamixelClass::setBaudrateDMXL(long baud){
      _baudrateDMXL = baud;
}

//Get the baudrate of the serial communication with DMXL
long DynamixelClass::getBaudrateDMXL(){
      return(_baudrateDMXL);
}

//Configure both ID and Baudrate of the servo. By changing the baudrate, the
    communications will restart automatically
//The next time time you call the servos you need to use the NEW baudrate
void DynamixelClass::configureServo(uint8_t ID, uint8_t newID, long baud){

      setID(ID, newID);
      setBD(newID, baud);

      //End communications and restart with new baudrate
      end();
      begin(baud, _directionPin);
}

//Set both angle limits.
void DynamixelClass::setAngleLimit(uint8_t ID, int CWLimit, int CCWLimit){
      sendWord(ID, EEPROM_CW_ANGLE_LIMIT_L, CWLimit, TWO_BYTES,
         DMXL_WRITE_DATA);
      sendWord(ID, EEPROM_CCW_ANGLE_LIMIT_L, CCWLimit, TWO_BYTES,
         DMXL_WRITE_DATA);
}

//Function to set both limits to 0. The servo is functioning in wheel mode
void DynamixelClass::setWheelMode(uint8_t ID, bool enable){
       if (enable){
             setAngleLimit(ID, 0, 0);}
       else{
             setAngleLimit(ID, 0, 4095);}
}
```

```
//Function to set the servo as joint mode. Equivalent to setWheelMode(ID,
   false)
void DynamixelClass::setJointMode(uint8_t ID){
       setAngleLimit(ID, 0, 4095);
}

//Function to set all gains
void DynamixelClass::setDIP(uint8_t ID, int gainD, int gainI, int gainP){
      setGainD(ID, gainD);
      setGainI(ID, gainI);
      setGainP(ID, gainP);
}

//Function to find the ID of the servo, if you have the correct baudrate.
   Assume begin() has been called
int DynamixelClass::findByBaudRate(long baudRate){
      int foundID;

      for(int i=0; i<254; i++){            //Search every ID possible
            if( (foundID=readID(i))!=-1 ){          //If we get anything but
                 a communication error, return the value
                   return(foundID);
            }
      }

      return(NO_SERVO_RESPONSE);            //Return error if nothing found
}

//Function to find the baudrate to communicate with the servo, if you have
   the correct ID. Assume begin() has NOT been called
int DynamixelClass::findByID(uint8_t id, uint8_t directionPin){
      int foundID;
      long roundedBaudRate;

      for(int i=0; i<=254; i++){
            roundedBaudRate = (2000000)/(i+1);         //Search every
               possible baudrate
            begin(roundedBaudRate, directionPin);        //begin
               communication

            if( (foundID=readID(id)) == id ){         //if we get the same
               ID as the one we assumed correct
                  return(readBD(id));            //return the baudrate
            }

            end();                //end communication
      }
```

```
      return(NO_SERVO_RESPONSE);            //If nothing found, return error
}

//Find the servo without having any information. Assume begin() has NOT been
    called
void DynamixelClass::findServo(uint8_t directionPin){
      int error;
      long roundedBaudRate;
      for(uint8_t i=0; i<=254; i++){         //Try every baudrate
        roundedBaudRate = (2000000)/(i+1);
        begin(roundedBaudRate, directionPin);

        for(uint8_t j=0; j<254; j++){         //Try every ID

         if( (error=readID(j)) != -1){          //If we get anything but an
             error
          //digitalWrite(led1, HIGH);
          printPC("Attempting␣ID:␣");
          printPC(j);
          printPC(",␣attempted␣baud␣rate␣is:␣");
          printPC(i);
          printPC(",␣and␣the␣returned␣baudrate␣is:␣");
          printlnPC(readBD(j));
         }
         else{
          printPC("Attempted␣ID:␣");
          printPC(j);
          printPC(",␣attempted␣baud␣rate␣is:␣");
          printlnPC(i);
          }
        }
        end();
       }
}

//Change time out period (waiting time for status package). Unit is [ms].
   The maximum value is 255
void DynamixelClass::changeTimeOut(uint8_t newTimeOut){
      TIME_OUT = newTimeOut;
}

//Change cool down period (time between sending commands). Unit is [ms]. The
    maximum value is 65,535 (i.e., 65.535 seconds)
void DynamixelClass::changeCoolDown(uint16_t newCoolDown){
      COOL_DOWN = newCoolDown;
}
```

```
//Print information about the servo. Use carefully since it uses the Serial
   port
void DynamixelClass::servoIntroduction(uint8_t ID){
      printlnPC("----------------------------------------");

      printPC("Hi, I am a servo model ");
      printPC(ID);
      printPC(" with Firmware version ");
      printPC(readFirmware(ID));
      printlnPC(", my ID is ");
      printPC(readID(ID));
      printPC(", communicating at a baudrate of ");
      printPC(readBD(ID));
      printPC(", with a RDT of ");
      printlnPC(readRDT(ID));

      printPC("CW limit set as ");
      printPC(readCWAngleLimit(ID));
      printPC(", CCW limit set as ");
      printlnPC(readCCWAngleLimit(ID));

      printPC("The temperature limit, lowest voltage limit, highest voltage
         limit, and max torque are: ");
      printPC(readTempLimit(ID));
      printPC(", ");
      printPC(readLowVoltageLimit(ID));
      printPC(", ");
      printPC(readHighVoltageLimit(ID));
      printPC(", ");
      printlnPC(readMaxTorque(ID));

      printPC("My Status return level, Alarm LED, and shutdown alarm
         settings are: ");
      printPC(readSRL(ID));
      printPC(", ");
      printPC(readAlarmLED(ID));
      printPC(", ");
      printlnPC(readShutdownAlarm(ID));

      printPC("The multi-turn offset setting and resolution divider are: ");
      printPC(readMultiTurnOffset(ID));
      printPC(", ");
      printlnPC(readResolutionDivider(ID));

      printPC("The DIP gains are: ");
      printPC(readGainD(ID));
      printPC(", ");
      printPC(readGainI(ID));
```

```
      printPC(",␣and␣");
      printlnPC(readGainP(ID));

      printPC("The␣value␣of␣moving␣speed␣is:␣");
      printlnPC(readMovingSpeed(ID));
      printPC("The␣value␣of␣torque␣limit␣(goal␣torque)␣is:␣");
      printlnPC(readTorqueLimit(ID));

      printPC("My␣present␣load␣is:␣");
      printPC(readLoad(ID));
      printPC("I␣am␣operating␣at␣a␣voltage␣of␣");
      printPC(readVoltage(ID));
      printPC("␣and␣a␣temperature␣of␣");
      printlnPC(readTemperature(ID));

      printPC("Is␣there␣a␣function␣waiting␣to␣be␣executed␣in␣Registered?:␣")
         ;
      printlnPC(registeredStatus(ID));

      printPC("Is␣a␣moving␣(goal␣position)␣command␣being␣executed?:␣");
      printlnPC(moving(ID));

      printlnPC("----------------------------------------");
}

//--------------------Multi-compatibility functions-------------------------

void DynamixelClass::sendData(uint8_t val){

      #if (PLATFORM_ID==88) || defined(SPARK)
            Serial1.write(val);        //For Duo or Photon
      #else
            Serial1.write(val);        //For Leonardo
      #endif
}

void DynamixelClass::sendDataBuff(uint8_t* buff, uint8_t len){

      #if (PLATFORM_ID==88) || defined(SPARK)
            Serial1.write(buff, len);      //For Duo or Photon
      #else
            Serial1.write(buff, len);      //For Leonardo
      #endif
}

// Check Serial Data Available
int DynamixelClass::availableData(void){
```

```
      #if (PLATFORM_ID==88) || defined(SPARK)
            return Serial1.available();
      #else
            return Serial1.available();
      #endif
}

// Read Serial Data
uint8_t DynamixelClass::readData(void){

      #if (PLATFORM_ID==88) || defined(SPARK)
            return Serial1.read();
      #else
            return Serial1.read();
      #endif
}

// Peek Serial Data
uint8_t DynamixelClass::peekData(void){

      #if (PLATFORM_ID==88) || defined(SPARK)
            return Serial1.peek();
      #else
            return Serial1.peek();
      #endif
}

// Begin Serial Comunication
void DynamixelClass::beginCom(long speed){

      #if (PLATFORM_ID==88) || defined(SPARK)
            Serial1.begin(speed);
      #else
            Serial1.begin(speed);
      #endif
}

// End Serial Comunication
void DynamixelClass::endCom(void){

      #if (PLATFORM_ID==88) || defined(SPARK)
            Serial1.end();
      #else
            Serial1.end();
      #endif
}

// Wait until data has been written
```

```
void DynamixelClass::serialFlush(void){

      #if (PLATFORM_ID==88) || defined(SPARK)
            Serial1.flush();
      #else
            Serial1.flush();
      #endif
}

// Macro for Timing. Delay milliseconds
void DynamixelClass::delayms(unsigned int ms){

      #if (PLATFORM_ID==88) || defined(SPARK)
            delay(ms);
      #else
            delay(ms);
      #endif
}

// Macro for Timing. Delay Microseconds
void DynamixelClass::delayus(unsigned int us){

      #if (PLATFORM_ID==88) || defined(SPARK)
            delayMicroseconds(us);
      #else
            delayMicroseconds(us);
      #endif
}

DynamixelClass Dynamixel;
```

Index

Bibliography

[1] Y. Yamamoto and X. Yun, "Coordinating locomotion and manipulation of a mobile manipulator", *IEEE Trans. Autom. Control*, vol. 39, no. 6, pp. 1326–1332, 1994, ISSN: 0018-9286.

[2] K. Nagatani, T. Hirayama, A. Gofuku, and Y. Tanaka, "Motion planning for mobile manipulator with keeping manipulability", in *Proc. IEEE Int. Conf. Intelligent Robots and Syst. (IROS 2002)*, vol. 2, 2002, 1663–1668 vol.2.

[3] R. Murphy, "Trial by fire [rescue robots]", *IEEE Robot. Autom. Mag.*, vol. 11, no. 3, pp. 50–61, 2004, ISSN: 1070-9932.

[4] A. Davids, "Urban search and rescue robots: From tragedy to technology", *IEEE Intell. Syst.*, vol. 17, no. 2, pp. 81–83, 2002, ISSN: 1541-1672.

[5] S. Hirose and E. F. Fukushima, "Snakes and strings: New robotic components for rescue operations", *Int. J. Robotics Research*, vol. 23, no. 4-5, pp. 341–349, 2004. DOI: 10.1177/0278364904042202. eprint: http://ijr.sagepub.com/content/23/4-5/341.full.pdf+html. [Online]. Available: http://ijr.sagepub.com/content/23/4-5/341.abstract.

[6] D. Roa and K. Melo, "Mechanical stability margin for scouting poses in modular snake robots", in *IEEE Int. Symposium on Safety, Security, and Rescue Robotics (SSRR 2016)*, 2016, pp. 182–188. DOI: 10.1109/SSRR.2016.7784296.

[7] S. Ma, "Analysis of creeping locomotion of a snake-like robot", *Advanced Robotics*, vol. 15, no. 2, pp. 205–224, 2001.

[8] M. Saito, M. Fukaya, and T. Iwasaki, "Serpentine locomotion with robotic snakes", *IEEE Control Syst. Mag.*, vol. 22, no. 1, pp. 64–81, 2002, ISSN: 1066-033X.

[9] P. Liljebäck, K. Pettersen, Ø. Stavdahl, and J. Gravdahl, "A simplified model of planar snake robot locomotion", in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS 2010)*, 2010, pp. 2868–2875.

[10] S. Ma and N. Tadokoro, "Analysis of creeping locomotion of a snake-like robot on a slope", *Autonomous Robots*, vol. 20, no. 1, pp. 15–23, 2006.

[11] R. L. Hatton and H. Choset, "Sidewinding on slopes", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA 2010)*, IEEE, 2010, pp. 691–696.

[12] S. Ma, Y. Ohmameuda, and K. Inoue, "Dynamic analysis of 3-dimensional snake robots", in *Proc. IEEE Int. Conf. Intelligent Robots and Syst. (IROS 2004)*, IEEE, vol. 1, 2004, pp. 767–772.

[13] P. Liljebäck, Ø. Stavdahl, and K. Y. Pettersen, "Modular pneumatic snake robot: 3d modelling, implementation and control", *Modeling, Identification and Control*, vol. 29, no. 1, pp. 21–28, 2008.

[14] R. L. Hatton, R. A. Knepper, H. Choset, D. Rollinson, C. Gong, and E. Galceran, "Snakes on a plan: Toward combining planning and control", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA 2013)*, 2013, pp. 5174–5181.

[15] J. S. Pettinato and H. E. Stephanou, "Manipulability and stability of a tentacle based robot manipulator", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA '89)*, IEEE, 1989, pp. 458–463.

[16] G. Chirikjian and J. Burdick, "The kinematics of hyper-redundant robot locomotion", *IEEE Trans. Robot. Autom.*, vol. 11, no. 6, pp. 781–793, 1995, ISSN: 1042-296X.

[17] Z. Wang, S. Ma, B. Li, and Y. Wang, "A unified dynamic model for locomotion and manipulation of a snake-like robot based on differential geometry", *Science China Information Sciences*, vol. 54, no. 2, pp. 318–333, 2011.

[18] G. Salvietti, H. Zhang, J Gonzalez-Gomez, D. Prattichizzo, and J. Zhang, "Task priority grasping and locomotion control of modular robot", in *Proc. IEEE Int. Conf. Robotics and Biomimetics (ROBIO 2009)*, IEEE, 2009, pp. 1069–1074.

[19] S. Nansai, M. R. Elara, and M. Iwase, "Dynamic hybrid position force control using virtual internal model to realize a cutting task by a snake-like robot", in *Proc. IEEE Int. Conf. Biomedical Robotics and Biomechatronics (BioRob 2016)*, IEEE, 2016, pp. 151–156. DOI: 10.1109/BIOROB.2016.7523614.

[20] P. Liljebäck, K. Y. Pettersen, Ø. Stavdahl, and J. T. Gravdahl, "Hybrid modelling and control of obstacle-aided snake robot locomotion", *Robotics, IEEE Transactions on*, vol. 26, no. 5, pp. 781–799, 2010.

[21] C. Holden and Ø. Stavdahl, "Optimal static propulsive force for obstacle-aided locomotion in snake robots", in *Proc. IEEE Int. Conf. Robotics and Biomimetics (ROBIO 2013)*, 2013, pp. 1125–1130.

[22] C. Holden, Ø. Stavdahl, and J. T. Gravdahl, "Optimal dynamic force mapping for obstacle-aided locomotion in 2d snake robots", in *Proc. IEEE Int. Conf. Intelligent Robots and Syst. (IROS 2014)*, 2014, pp. 321–328.

[23] Z. Y. Bayraktaroglu and P. Blazevic, "Understanding snakelike locomotion through a novel push-point approach", *Journal of dynamic systems, measurement, and control*, vol. 127, no. 1, pp. 146–152, 2005.

[24] F. Reyes, W. Tang, and S. Ma, "Using a planar snake robot as a robotic arm taking into account the lack of a fixed base: Feasible region", in *Proc. IEEE Int. Conf. Intelligent Robots and Syst. (IROS 2015)*, 2015, pp. 956–962.

[25] F. Reyes and S. Ma, "Snake robots in contact with the environment: Influence of the configuration on the applied wrench", in *Proc. IEEE Int. Conf. Intelligent Robots and Syst. (IROS 2016)*, 2016, pp. 3854–3859.

[26] ——, "Snake robots in contact with the environment - influence of the friction on the applied wrench", in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS 2017)*, 2017, pp. 5790–5795. DOI: 10.1109/IROS.2017.8206471.

[27] ——, "Studying slippage on pushing applications with snake robots", in *In: Proc. Int. Conf. Real-time Computing and Robotics (RCAR 2017)*, 2017.

[28] O. Khatib, "Inertial properties in robotic manipulation: An object-level framework", *Int. J. Robotics Research*, vol. 14, no. 1, pp. 19–36, 1995. DOI: 10.1177/027836499501400103. eprint: http://dx.doi.org/10.1177/027836499501400103. [Online]. Available: http://dx.doi.org/10.1177/027836499501400103.

[29] F. Reyes and S. Ma, "On planar grasping with snake robots: Form-closure with enveloping grasps", in *Proc. IEEE Int. Conf. Robotics and Biomimetics (ROBIO 2014)*, 2014, pp. 556–561.

[30] ——, "Modeling of snake robots oriented towards grasping and interaction with the environment", in *In: Proc. Int. Conf. Real-time Computing and Robotics (RCAR 2015)*, not published, 2015.

[31] F. Reyes, H. Matsumoto, and S. Ma, "Design and implementation of modular and parametric 3d printed snake robot", *The Robotics and Mechatronics Conference (ROBOMECH 2017)*, vol. 2017, 2A2–A11, 2017. DOI: 10.1299/jsmermd.2017.2A2-A11. [Online]. Available: https://www.jstage.jst.go.jp/article/jsmermd/2017/0/2017_2A2-A11/_article/-char/ja.

[32] F. Reyes and S. Ma, "Studying slippage on pushing applications with snake robots", *Robotics and Biomimetics*, vol. 4, no. 1, p. 9, 2017, ISSN: 2197-3768. DOI: 10.1186/s40638-017-0065-3. [Online]. Available: https://doi.org/10.1186/s40638-017-0065-3.

[33] W. Blajer, "A geometrical interpretation and uniform matrix formulation of multibody system dynamics", *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 81, no. 4, pp. 247–259, Apr. 1, 2001, ISSN: 1521-4001. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/1521-4001(200104)81:4<247::AID-ZAMM247>3.0.CO;2-D/abstract (visited on 01/03/2016).

[34] R. Featherstone, *Rigid body dynamics algorithms*. USA: Springer US, 2014.

[35] R. M. Murray, S. S. Sastry, and Z. Li, *A Mathematical Introduction to Robotic Manipulation*, 1st. Boca Raton, FL, USA: CRC Press, Inc., 1994, ISBN: 0849379814.

[36] R. Featherstone, *Rigid Body Dynamics Algorithms*. Boston, MA: Springer US, 2008, ISBN: 978-0-387-74314-1 978-1-4899-7560-7. [Online]. Available: http://link.springer.com/10.1007/978-1-4899-7560-7 (visited on 10/29/2015).

[37] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. New York, USA: Springer-Verlag Inc., 2008, ISBN: 978-3-540-23957-4.

[38] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, 1st. London: Springer Publishing Company, Incorporated, 2008, ISBN: 1846286417, 9781846286414.

[39] D. Prattichizzo and A. Bicchi, "Dynamic analysis of mobility and graspability of general manipulation systems", *IEEE Trans. Robot. Autom.*, vol. 14, no. 2, pp. 241–258, 1998, ISSN: 1042-296X.

[40] A. Bicchi, C. Melchiorri, and D. Balluchi, "On the mobility and manipulability of general multiple limb robots", *IEEE Trans. Robot. Autom.*, vol. 11, no. 2, pp. 215–228, 1995, ISSN: 1042-296X.

[41] P. Lötstedt, "Mechanical systems of rigid bodies subject to unilateral constraints", *SIAM Journal on Applied Mathematics*, vol. 42, no. 2, pp. 281–296, 1982. DOI: 10.1137/0142022.

eprint: https://doi.org/10.1137/0142022. [Online]. Available: https://doi.org/10.1137/0142022.

[42] P. Liljebäck, K. Y. Pettersen, Ø. Stavdahl, and J. T. Gravdahl, *Snake robots: modelling, mechatronics, and control*. Springer Science & Business Media, 2012.

[43] J Gray, "The mechanism of locomotion in snakes", *J. Exp. Biol.*, vol. 23, no. 2, pp. 101–120, 1946.

[44] F. Matsuno and H. Sato, "Trajectory tracking control of snake robots based on dynamic model", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA 2005)*, 2005, pp. 3029–3034.

[45] K. Watanabe, M. Iwase, S. Hatakeyama, and T. Maruyama, "Control strategy for a snake-like robot based on constraint force and verification by experiment", *Advanced Robotics*, vol. 23, no. 7-8, pp. 907–937, 2009.

[46] H. Date, Y. Hoshi, and M. Sampei, "Locomotion control of a snake-like robot based on dynamic manipulability", in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS 2000)*, IEEE, vol. 3, 2000, pp. 2236–2241.

[47] F. Matsuno and K. Mogi, "Redundancy controllable system and control of snake robots based on kinematic model", in *Proc. IEEE Int. Conf. on Decision and Control (CDC 2000)*, IEEE, vol. 5, 2000, pp. 4791–4796.

[48] P. Grinfeld, *Introduction to tensor analysis and the calculus of moving surfaces*. London: Springer, 2013.

[49] G. Liu and Z. Li, "A unified geometric approach to modeling and control of constrained mechanical systems", *IEEE Trans. Robot. Autom.*, vol. 18, no. 4, pp. 574–587, 2002, ISSN: 1042-296X.

[50] T. Inoue and S. Hirai, *Mechanics and Control of Soft-fingered Manipulation*. London: Springer-Verlag London, 2009, ISBN: 978-1-84800-980-6.

[51] Y. Chen, "Equations of motion of constrained mechanical systems: Given force depends on constraint force", *Mechatronics*, vol. 9, no. 4, pp. 411 –428, 1999, ISSN: 0957-4158. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957415898000531.

[52] S. Hirose, *Biologically Inspired Robots: Snake-like Locomotors and Manipulators*. Oxford University Press, 1993.

[53] A. Kuwada, S. Wakimoto, K. Suzumori, and Y. Adomi, "Automatic pipe negotiation control for snake-like robot", in *Proc. IEEE Int. Conf. Advanced Intelligent Mechatronics (AIM 2008)*, IEEE, 2008, pp. 558–563.

[54] A. Shapiro, A. Greenfield, and H. Choset, "Frictional compliance model development and experiments for snake robot climbing", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA 2007)*, IEEE, 2007, pp. 574–579.

[55] A. Greenfield, A. A. Rizzi, and H. Choset, "Dynamic ambiguities in frictional rigid-body systems with application to climbing via bracing", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA 2005)*, IEEE, 2005, pp. 1947–1952.

[56] Y. Guan, L. Jiang, H. Zhu, W. Wu, X. Zhou, H. Zhang, and X. Zhang, "Climbot: A bio-inspired modular biped climbing robot—system development, climbing gaits, and experiments", *Journal of Mechanisms and Robotics*, vol. 8, no. 2, p. 021 026, 2016.

[57] K. T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing", in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS 2016)*, 2016, pp. 30–37. DOI: 10.1109/IROS.2016.7758091.

[58] P. Liljebäck, K. Pettersen, Ø. Stavdahl, and J. Gravdahl, "A review on modelling, implementation, and control of snake robots", *Robot. Auto. Sys.*, vol. 60, no. 1, pp. 29 –40, 2012, ISSN: 0921-8890.

[59] J. C. Trinkle, J. M. Abel, and R. P. Paul, "Enveloping, frictionless, planar grasping", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA '87)*, Citeseer, 1987.

[60] A. Bicchi, "On the Closure Properties of Robotic Grasping", *International Journal of Robotics Research*, vol. 14, pp. 319–334, 1995.

[61] F. E. Udwadia and R. E. Kalaba, "A new perspective on constrained motion", English, *Proceedings: Math. and Physical Sciences*, vol. 439, no. 1906, pp. 407–410, 1992, ISSN: 09628444. [Online]. Available: http://www.jstor.org/stable/52227.

[62] E. Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA 2014)*, 2014, pp. 6054–6061.

[63] R. Suárez, J. Cornella, and M. R. Garzón, *Grasp quality measures*. Institut d'Organització i Control de Sistemes Industrials, 2006.

[64] D. Prattichizzo, M. Malvezzi, M. Gabiccini, and A. Bicchi, "On the manipulability ellipsoids of underactuated robotic hands with compliance", *Robotics and Autonomous Systems*, vol. 60, no. 3, pp. 337–346, 2012, ISSN: 0921-8890. DOI: 10.1016/J.ROBOT.2011.07.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889011001461.

[65] X. Markenscoff, L. Ni, and C. H. Papadimitriou, "The geometry of grasping", *Int. J. Robot. Res.*, vol. 9, no. 1, pp. 61–74, 1990.

[66] A. Rodriguez, M. T. Mason, and S. Ferry, "From caging to grasping", *Int. J. Robot. Res.*, vol. 31, no. 7, pp. 886–900, 2012.

[67] J. T. Y. Wen and L. S. Wilfinger, "Kinematic manipulability of general constrained rigid multibody systems", *IEEE Trans. Robot. Autom.*, vol. 15, no. 3, pp. 558–567, Jun. 1999, ISSN: 1042-296X. DOI: 10.1109/70.768187.

[68] B. Bayle, J. Y. Fourquet, and M. Renaud, "Manipulability analysis for mobile manipulators", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA 2001)*, vol. 2, 2001, 1251–1256 vol.2. DOI: 10.1109/ROBOT.2001.932782.

[69] K. van den Doel and D. K. Pai, "Constructing performance measures for robot manipulators", in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA 1994)*, 1994, 1601–1607 vol.2.

[70] S. Patel and T. Sobh, "Manipulator performance measures-a comprehensive literature survey", *Journal of Intelligent & Robotic Systems*, vol. 77, no. 3-4, pp. 547–570, 2015.