Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Extending the Vereshchagin hybrid dynamic solver to mobile robots

Sushma Devaramani

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul G. Plöger
Sven Schneider

January 2019

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Sushma Devaramani

Abstract

The objective of the project is to extend and apply the Vereshchagin hybrid dynamic solver to mobile robots. A typical execution of mobile robot tasks involves navigation from one point to another by effectively avoiding obstacles. In autonomous systems, there are various algorithms employed to implement collision avoidance. These approaches follow velocity-based control scheme, which primarily aims at ignoring physical contact with the objects around the robot. However, if the situation demands physical contact robot must not cause any damage to the environment. However, when the robot comes across an obstacle unexpectedly, the velocity control strategy fails. The reason for failure is that the control scheme cannot instantly detect the object and control the robot motions. Therefore, there is a need to include safety constraints which the robot must handle while executing its functions. The issue of handling safety constraints has been addressed in robot manipulators for ages since they are continuously involved in manipulating the objects in the world. Additionally, the diversity of robot motion tasks has led to the development of (constrained) task control methodologies with origins in force control, humanoid robot control, mobile manipulator control, visual servoing, etc. The sequence of tasks such as pick and place operations in manipulators are executed through task specification strategy, where each of the associated task constraints is modeled. Nevertheless, there is no specific task specification approach employed in mobile robots. In robot manipulators, there are several software frameworks, algorithms and dynamic solvers employed to realize the task constraints instantly and efficiently. The Popov-Vereshchagin solver is one such dynamic solvers practiced by manipulators. The Vereshchagin is a significant algorithm for the posture control of mobile manipulators and humanoid robots since such tasks typically require specifications of motion and/or force constraints on the end-effectors and other segments. Additionally, the Vereshchagin solver can be applied to closed as well as open kinematic chains. Since the wheels and base of

the mobile robot can be modeled as a closed kinematic chain, the solver can be extended and applied to mobile robots.

Acknowledgements

Thanks to ....

Contents

List of Figures

List of Tables

Acronyms

ABA	-	Articulated-Body Algorithm
CRBA	-	Composite Rigid Body Algorithm
iTaSC	-	instantaneous Task Specification and Control
KDL	-	Kinematic and Dynamics Library
OCP	-	Optimal Control Problem
RNEA	-	Recursive Newton-Euler Algorithm
SoT	-	Stack of Tasks
TFF	-	Task Frame Formalism
VFF	-	Vector Force Field
WBC	-	Whole Body Control
WBOSC	-	Whole-Body Operational Space Control

List of symbols

M	Inertia matrix that maps between joint space domain and force domain
q	Joint position vector
$\dot{q}$	Joint velocity vector
$\ddot{q}$	Joint acceleration vector
f_c	Joint space constraint forces
$\hat{b}(q, \dot{q})$	Bias acceleration over second order derivative of holonomic position constraint
τ_a	Input forces
τ_c	Constraint forces
$C(q, \dot{q})$	Bias forces
$\ddot{X}$	Cartesian acceleration
H_i	Inertial matrix of link i
$F^T_{bias,i}$	Vector comprising of Coriolis and centrifugal forces
F_N	Cartesian space constraint force vector applied on segment N
d	Moment of rotor inertia
A_N	Linear constraint matrix of order 6 x m where m is the number of constraints on a segment
b_N	Acceleration energy (force times acceleration)

1

Introduction

Safety is one of the critical factors to be considered when designing robotic systems in human environments [51]. The robotic engineers and researchers from an extended period, have focused on the safety of robots and its workspace. The growing application of the two main classes of robots, i.e., manipulators and mobile robots in diverse fields adds to the necessity for safety.

Robot manipulators are widely employed in an industrial environment. As manipulators are bulky and dangerous, the tasks are confined to a closed environment, away from humans. However, recently, the advancement in the field of manipulators has contributed to a safe interaction with humans. The increasing complexity in tasks has led to never-ending research in the collision handling systems. For instance, a robotic arm performing pick and place operation in a structured environment has to plan and execute the task safely by achieving dynamic collision avoidance. Additionally, there are various constraints imposed by the task specification. One such constraint would be to place the object vertically on the table without damaging the object and the workspace. Likewise, many such constraints are imposed as the complexity of tasks increases. There are software frameworks and dynamic solvers targeted to realize the constraints in real-time.

Consequently, in the field of autonomous mobile robots, safe navigation is the crucial goal [27]. Due to their ability to navigate, mobile robots are often employed in applications such as logistics, security and defense, inspection and maintenance, cleaning, agriculture and many more. Typically navigating in populated environments, the mobile robot performs a task under changing external circumstances. Therefore, the robots must plan dynamically to respond to such unforeseen situations [27].

YET TO WRITE..............

1.1 Motivation

The robot navigation has been implemented effectively by many approaches. However, these methods often perform obstacle avoidance [19] [8]. And the robot motions are controlled at velocity-level. In some circumstances, the objectives demand force/acceleration constraints if the robot is obliged to come in contact with the environment. The traditional velocity-based control cannot handle the constraints in force/acceleration level. Hence there is a necessity to manage these constraints in mobile robots. In contrast to mobile robots, the need for continuous physical interaction with the environment has already been recognized for several decades in the manipulators. This field is well researched in robotic arms that manipulate objects. The arm/joint parameters are bound by specific force constraints [9]. Specifically, the end-effector joints are limited by allowable force on the object. For instance, consider a pick and place scenario, where the arm has to grasp a fragile glass and place it on a workbench. Here, the end-effector has to grip with a specific force such that the glass neither breaks nor slips out. Additionally, the task might impose multiple constraints, such as the end-effector must place the object perpendicular to the plane by applying limited forces. The arm must satisfy these dynamic constraints. The controller supervises these constraints at that instant of time. Besides, many such task constraints are imposed and hence dynamic solvers are used to realize them instantly.

The Vereshchagin solver is one such dynamic solver that can handle the requirements presented above in robotic manipulators. The aim of the project is to extend and apply this solver to mobile robots.

1.2 Challenges and Difficulties

YET TO BE WRITTEN!!

1.3 Problem Statement

The robot manipulators are extensively involved in physical interactions with the objects in the environment. There are various software frameworks and dynamic solvers to manage the task constraints while performing manipulation tasks. However, the mobile robots do not exhibit direct physical contact with the world unless in two circumstances,

- When the robot comes across an obstacle unexpectedly;
  Use case: Consider an autonomous system navigating from point A to point B by avoiding obstacles. When the robot has to turn around a corner, it is unaware of any approaching obstacles. In such situations, the base must exhibit motions with limited force. Even if the robot comes in contact with the obstacle, it should not harm the environment.
- When the robot task involves contact with an object;
  1. **Use case:** Consider a multi-robot system performing logistic tasks in an industrial environment, where a robot has to join itself to another through some means (e.g., a hook). For this purpose, the robot initially has to align and come in contact with the other robot physically to connect itself. In this example, the task demands constraints such as safe alignment with limited acceleration.
  2. **Use case:** Consider a wall alignment problem. The usual procedure is to detect the wall, and the mounted sensors continuously compute the distance values from the wall. Based on these values, the robot adjusts its position. In spite of this traditional method, the project presents an approach to exploit the obstacle. If a virtual force is pushing the robot towards the wall, and at one point it comes in contact with the wall. There is an acceleration constraint when it tries to move further. The solver equipped in the article utilizes this constraint to align the robot to the wall.

The project addresses the safety constraints in the situations as explained in use cases. The project also addresses the issue regarding task specification for mobile bases. Generally, the manipulators involve a task specification strategy to fulfill the sequence of tasks. These tasks impose several constraints on the robot actions. Many software frameworks handle these constraints at the task level. However, in the field of mobile robots, there is no practical implementation of task specification approach. Below is a use case that depicts why task specification procedure would be helpful for mobile bases.

- **Use case:** A mobile robot is performing logistic functions in a hospital environment. The task requirement is to carry objects to a destination. Limited velocity and forces constrain the robot motions. Additionally, the robot must drive inside a specified boundary. The robot must effectively be able to handle them instantly. The project presents a similar approach to task specification for mobile bases.
  The project seeks to solve the issue of handling the constraints arising from multiple tasks.

2

State of the Art

The current state of the art focuses on various approaches to implement complex robot tasks involving robust motions and complex motion primitives. As mentioned earlier (section 1), the task requirements impose explicit constraints on robot motions. These constraints indicate the desired force or motion to be executed by the robot. It is imperative to consider the dynamic properties of the system to realize these constraints instantaneously and execute the optimal motions. In this section, current state of the art relating to robot dynamic algorithms, task specification formalisms and dynamic solvers are summarized briefly.

2.1 Robot dynamics algorithms

Robot dynamics deals with the relationship between applied force and produced accelerations in the system [25]. The robot dynamics algorithms refer to numerical computations of quantities associated with dynamics. It is well known that the robot dynamics problem is of two types - forward and inverse dynamics. The forces applied on any rigid body produces acceleration in the direction of applied force, this is termed as *forward dynamics*. The equation used to solve forward dynamics problem is given by [25],

$$FD \rightarrow M(q)^{-1}(\tau - C(q, \dot{q})) = \ddot{q} \tag{2.1}$$

where, $M(q)$ stands for inertia matrix represented in joint space and is a function of joint position (q). τ denotes the applied force and C is the Centrifugal and

Coriolis forces acting on the system. The *inverse dynamics* deals with computation of forces required to produce the desired acceleration. The equation used to solve inverse dynamics problem can be formulated as [57] [25],

$$ID \rightarrow M(q)\ddot{q} + C(q, \ddot{q}) = \tau \tag{2.2}$$

The above equation is also termed as *dynamic equation of motion* for rigid body system (further explanation can be found in appendix A). There are several types of robots such as manipulators, mobile robots, aerial robots etc, which are composition of rigid bodies. In this project, to simplify the analysis of robot dynamics, *Spatial notations* are used to represent the system and follows the convention as used in Featherstone [25]. The Spatial notions include 6D vectors describing six degrees of freedom of a single rigid body.

The applications of forward dynamics can be found mainly in simulation, whereas, inverse dynamics is applied for motion control system [24]. However, there are different robot task definitions that requires combination of forward and inverse dynamics. Specifically for applications involving *posture control* (humanoid robots and manipulators), the robot must realize the motion and force constraints instantaneously as imposed by the task requirements. The basic algorithms to solve each of the dynamics problem are listed below,

1. *Forward dynamics*

   - *Composite Rigid Body Algorithm (CRBA)* [58]: For the given link length, $n < 9$, this method is an efficient algorithm than ABA, to compute forward dynamics [26].
   - *Articulated-Body Algorithm (ABA)*: The method considers whole system as articulated body and computes the forward dynamics. It has *O(n)* computational complexity.

2. *Inverse dynamics*

   - *Recursive Newton-Euler Algorithm (RNEA)*: The algorithm is applied to calculate inverse dynamics of a general kinematic tree [26]. It involves two passes - *outward* and *inward*. In outward pass, *velocity* and *acceleration* quantities are computed from base to the leaves and *joint forces* are computed from leaves to the root during inward pass [25].

3. *Hybrid dynamics*

   - *Articulated-Body Hybrid Dynamics Algorithm* - An articulated-body algorithm applied to perform combined forward and inverse-dynamics.
   - *Popov-Vereshchagin Hybrid Dynamic Algorithm* - applied mainly to kinematic chain to solve hybrid dynamics problem (further description is provided in Chapter 3).

All these algorithms can also be extended to *floating bases*, by converting floating-base system to fixed-base system [25]. Here, floating base is a rigid-body system, whose base is not fixed. Examples of floating bases are, mobile robots, mobile manipulators etc.

A robotic system is subjected to *constraints*, which can either be imposed by environmental contacts (*physical constraints*) or task requirements (*artificial constraints*). Considering these constraints, the dynamic equation of motion is reformulated as [50],

$$M(q)\ddot{q} + C(q, \ddot{q}) = \tau - \tau_c \tag{2.3}$$

where, τ_c is constraint force vector and is subjected to *holonomic position constraint*, $h(q) = 0$. However, the obtained equation is not optimal. Chapter 3 explains solver that computes optimal solution to the equation 2.3.

Open source libraries available to implement Rigid body algorithm............

2.2 Software Frameworks

This section discusses primitive software frameworks implemented in the area of robot manipulators to handle the constraints originating from task requirements.

2.2.1 Task Frame Formalism (TFF)

The manipulator actions are constrained due to the constant interaction with the environment. This constrained motion is also entitled as compliant motion [35]. Task Frame Formalism is an intuitive approach that executes desired actions (force-controlled actions) compatible with constraints imposed by the task [14]. The method is also called a *Compliance frame formalism.* A TFF frame is represented as follows [35],

$$\mathcal{TF} := \left\{\bar{\mathcal{P}}, RF, ANC\right\} \tag{2.4}$$

Here, $\mathcal{TF}$ refers to *Task Frame*, which describes one frame respective to another in task definition. In the above notation (2.4), $\bar{\mathcal{P}}$ is pose of *task frame* expressed in *reference frame* (RF). ANC is the *anchor* that rigidly sets TF onto another frame. To specify any compliant motion following information is required [19],

- Task frame *position* and *orientation*
- Specifying position and force controlled directions
- Target position and force represented in task frame

The main feature of the approach is to execute a sequence of manipulation tasks (specifically, atomic actions) maintaining the desired contact force [39] [47]. The formalism is independent of the control aspects and uses *task-oriented* concept, which means that the method enables a distinct task specification [14]. TFF is also used for motion constraint modeling and identification of uncertainty in compliant actions. The main drawback of TFF is that, it cannot handle changing motion constraints [16].

2.2.2 Whole-Body Operational Space Control (WBOSC)

Operational Space Formulation (OSF) was introduced to specify and control whole-body motion in robots [31] [32]. The framework primarily analyses and controls manipulator motions in accordance to dynamic features of end-effectors. This approach was further extended to WBOSC [33], [17], [49], [48], [34].

A *task-oriented*framework introduced to specify and control whole-body motion was *operational body formulation* For redundant robots, the task description might involve combination of different coordinates and

2.2.3 Stack of Tasks (SoT)

The *Stack of tasks* introduces a hierarchy of tasks to control redundant robots (manipulators and humanoid robots). The approach was presented in [37], [36], [44]. Generally, the tasks description specifies motion with bilateral constraints, given by [36],

$$e = s - s^* \tag{2.5}$$

where e is the (error) difference between actual (s) and desired (s^*) feature. This error function must converge to 0. Additionally, there are tasks that requires description of unilateral constraints (inequality constraints), which are represented by $e \leq 0$. Example of such tasks are obstacle avoidance [38], robot joint limits [18] or singularity avoidance [41]. Considering both constraints, SoT prioritizes the tasks to better achieve desired motion, i.e., the lower priority tasks are projected in free motion space of higher priority tasks. However, the approach cannot handle unilateral constraints regarding contact forces. Hence, different methods were introduced as an extension to current formalism [45] [46].

2.2.4 iTaSC - instantaneous Task Specification and Control

iTaSC is a *constraint-based* approach introduced in [20], [53]. The increase in complex robot tasks has led to the development of a systematic framework that can provide instantaneous task specification parallelly dealing with geometric uncertainty. Previously discussed approaches: *TFF*, *whole-body control framework* and *Stack of Tasks* are based on the *task function approach.* These approaches execute relative motion between robot and its environment through controlled dynamic interactions. Furthermore, task requirements impose certain constraints on robot motions. These constraints are not modeled in task function approach. Hence, a generic framework is required to realize the task constraints and also model geometric uncertainties.

As an extension to *task frame* concept, iTaSC has introduced *feature frames*. A part of *feature frame* is based on *task frame* itself and the other part specifies task constraints. Additionally, to specify relative pose between objects, the authors have introduced *object frames*. As mentioned earlier, the approach deals with the geometric uncertainties, that are expressed using uncertainty coordinates. The uncertainties might originate from modeling errors or external geometric disturbances. These coordinates are further used for error reduction in task execution.

A Generic control scheme is presented by the authors in [20], [53],

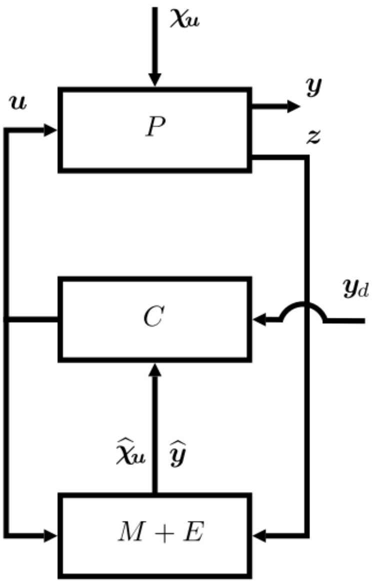

Figure 2.1: General control scheme of constraint-based approach (source: [20])

In the above figure, P is *plant*, that represent overall system (robot and its environment). The inputs to the system are desired control parameters such as joint positions, torques or velocities collectively represented by signal u and X_u representing geometric uncertainties. y is the system output variables and z are sensor measurements. As seen in figure (2.1), the input signal u is distributed between C and (M + E) blocks. Here, C represents *Control* block. There is another input to Control block, i.e., y_d, that represents desired values. The constraints imposed on system output y is converted to y_d. Other inputs to the controller include $\widehat{\mathcal{X}_u}$ and $\widehat{y}$ representing uncertainty and output estimates respectively. These estimates are produced from *model update* and *estimation* block (M + E) [20].

Initially, the approach failed to consider the unknown dynamic parameters (friction and stiffness) [20] and inequality constraints while computing robot motions. This deficit was further overlooked and authors extended the approach to compute the resultant motions as optimization problem [22] [21].

So far, the iTaSC framework has been implemented in various robotic systems [29], [55], [54]. There is an open-source software[1] available under Orocos project called iTaSC-Skill [4] that combines different iTaSC specifications. The software is also integrated in ROS. The software uses Bayesian Filtering Library (BFL) and KDL libraries to retrieve sensor data and representing virtual kinematic structure of robot.

One example of iTaSC application is PR2 co-manipulation (figure 2.2) that was presented at IROS 2011, San Fransisco, CA.

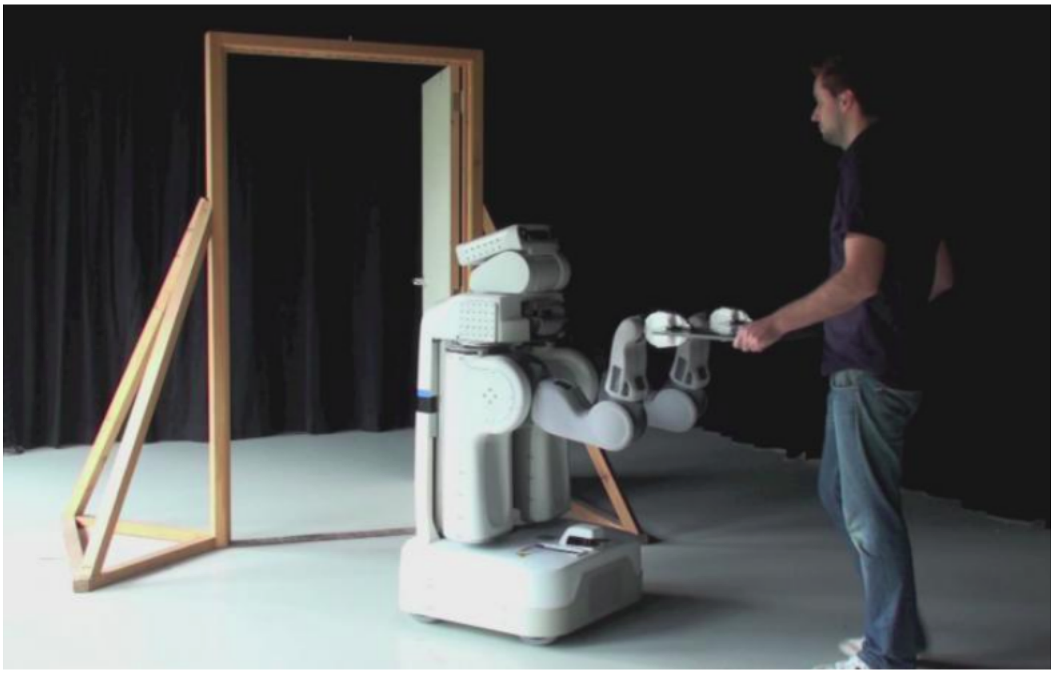

Figure 2.2: iTaSC demonstration on PR2

[1] iTaSC Open-source software available at https://gitlab.mech.kuleuven.be/rob-itasc

Given the set of task constraints such as head tracking, joint limits, force follow, maintaining the grippers parallel to each other and obstacle avoidance, the demonstration successfully implemented the task. The main drawback of the approach is that, the robot motions are controlled at *velocity-level* and considers only equality constraints [4].

2.3 Safe-navigation approaches

An autonomous mobile robot navigate from its current location to destination by avoiding obstacles along its way. An important consideration is the safety of robot and its environment. The researchers have introduced several methods to implement safe navigation [13], [6]. There are few methods that consider the dynamic model to control mobile robot motions [27], [8], [28], [12].

The literature [27] proposes *dynamic window approach* that implements an effective collision avoidance technique in mobile robots with synchronous drives. The method is derived from motion dynamics. In a populated and unknown environment, a mobile robot must react immediately to unforeseen obstacles. The author proposes that, to react to the unexpected obstacles or circumstances, the robot must dynamically re-plan so as to reach its destination. Here, the velocity search space is reduced, by considering the velocity commands achievable under dynamic constraints.

[27] This approach to obstacle avoidance is experimented on RHINO (robot equipped with synchronous drive). While driving through obstacle free area, RHINO navigates with highest velocity, and if an obstacle is detected, velocity of the robot is decreased by selecting suitable trajectories. Consider a situation where the robot is moving along a long corridor and there is an opening to the right. Now, the robot has to decide if it can take an immediate turn to right or not. These decisions are made by considering the robot dynamics. The robot will either take a sharp turn (only if robot has suitable velocity and acceleration) or tries to align to the wall by reducing heading angle. The main drawback of the approach is that in the experiment author assumes to have information about the

obstacle's relative locations and hence the approach is appropriate for proximity sensors. For different sensors, further assumptions have to be made.

The paper [8] presents kinematic and dynamic modeling of differential drive controller. When a virtual force is applied on the robot, considering the kinematic and dynamic constraints, the approach computes the velocities and accelerations compatible with the constraints. Here, the robot model itself is used as controller (also called as *reference model*) which also acts as motion generator. By providing a physical sense to controller parameters, they are adjusted to satisfy the dynamic behavior of the robot. This results in an automatic computation of desired velocities with no further tuning of robot parameters. The author presents a closed-loop control scheme depicting *reference model* as controller.

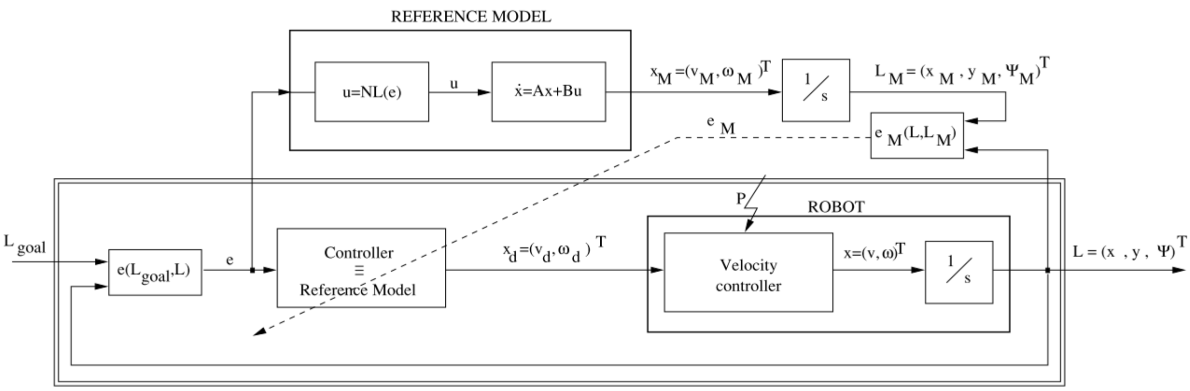

Figure 2.3: Reference model integrated in closed-loop control scheme(source: [8])

For a given navigation task, the *reference model* (above figure) computes the error (e_M) between desired and current location of robot. This error is further adjusted according to robot parameters. To use this reference model as controller, the method assumes to have robot's velocity control scheme that is fixed. If error calculated is 0 ($e_M = 0$), then the final control loop looks like the double-lined block. The error, $e_M \neq 0$ represents that the robot parameters (velocities and acceleration) are not within the scope. Then offline retuning of the controller parameters is done as a function of error e_M. The method do not consider the dynamic robot parameters, instead it fine tunes them to obtain desired robot motion in presence of kinematic and dynamic constraints. The paper also presents the experimentation, where, reactive navigation i.e., avoiding unknown obstacles to achieve safe navigation is executed using this *reference model*. The main drawback of the approach is that it implements velocity controller and assumes it to be fixed. However, In a dynamic environment, the controller must execute instantaneous robot motions to avoid unexpected collisions. This can be achieved at acceleration level rather than velocity-level

In the literature [12], the authors have proposed *Vector Force field* concept to implement real-time *collision avoidance* in mobile robots. The method is based on two abstractions, a) Representation of obstacles as *Certainty Grids*, b) Navigation based on *Potential Fields*. Combining these two abstractions and accounting to dynamic properties of mobile robots, VFF solves *local minimum trap* [30], [42].

The *local minimum trap* problem regards to the robot being trapped due to the occurrence of local minimum in the *potential field* approach and robot cannot proceed with further exploration. There are two algorithm presented in the approach. One to enhance the dynamic motion of robot and other to recover from *trap*. Additionally, the method considers the situation of collision avoidance in dynamic environment as wall-following problem, i.e., after encountering obstacle, the robot follows, while maintaining constant distance, along the contour of detected object until its reaches original course.

The approach [12] is tested on a mobile robot platform named CARMEL, in which the real time model was represented by *Certainty grids*. Combining this representation with *virtual potential field* resulted in clusters. The cluster with high likelihood is considered as an obstacle. Additionally, the possible trap conditions were eliminated by following a set of heuristic rules. However, the method fails to avoid unexpected obstacles, because motions controlled at velocity level cannot cope with instantaneous change in motion.

2.4 Limitations of previous work

The current state of the art discussed various software frameworks and safe-navigation approaches. The limitations of these methods with respect to the use cases explained in 1.3 are presented in this section.

The Software frameworks are currently deployed in manipulators. The key feature of these frameworks is that, they support various constraints originating from task specifications, to accomplish desired motion. It also provides a customized way of specifying and executing complex tasks. The utilities of these task specification frameworks have not been explored for mobile robots.

Furthermore, the safe navigation techniques currently implemented in mobile robots are discussed in 2.3. In these methods, the robot motions are controlled at *velocity-level*

YET TO EDIT!

above discussed safe-navigation approaches (in section 2.3), aims at achieving safe-navigation while considering the dynamic model of the robot. However, in all those methods, the robot motions are controlled at *velocity-level*. The use cases explained in section 1.3, need to control the robot motions at force and acceleration level.

3

Popov-Vereshchagin Hybrid Dynamics Solver

The use cases discussed in section 1.3, requires a way to control the robot motions such that the constraints are satisfied.

3.1 Solver Introduction

The Popov-Vereshchagin solver is a linear-time constrained hybrid-dynamic solver derived from one of the principles of mechanics - *Gauss principle of least constraints* [56], that formulates a "dynamically natural way" to solve the redundancy problem in manipulators [15]. The researches based on The main feature of the solver is that it computes desired acceleration of the robot in turn resolving the constraints originating from task requirements. Previous work on solver [23]

3.2 Solver Derivation

As mentioned in the introduction, the solver is derived from *Gauss principle of least constraints* At the basis, the principle states that "*Out of all the possible motions (accelerations) that are complied with the constraints of a system, a true motion (acceleration) is executed, which corresponds to minimum acceleration energy*". This true acceleration is the closest possible acceleration to an unconstrained system. Here, the solver computes the true acceleration of the kinematic chain by minimizing the *acceleration energy*.

As defined by the task requirements in a manipulator, various Cartesian acceleration constraints are imposed on one or more segments. Physically, these constraints are realized by forces exerted to limit the motion (acceleration) of segments in certain direction, which in turn produces acceleration energy.

The solver computes the solution to a constrained system that can be formulated as [50],

$$M(q)\ddot{q} + f_c = \tau_a(q) - C(q, \dot{q}) \tag{3.1}$$

The equation 3.1 is derived from the robot's dynamic motion model [50]. See the appendix section A for the complete explanation. Here $M(q)$ represents inertial matrix that maps from joint space ($\ddot{q}$) to force space (τ). The term f_c denotes constraint forces acting on the joints.

As previously mentioned, the solver minimizes the Gauss function to compute the true motion and resolves the redundancy problem in manipulators. It is given by [56],

$$\mathcal{Z} = \min_{\ddot{q}} \left\{ \sum_{i=0}^{N} \frac{1}{2} \ddot{X}_i^{\,T} H_i \ddot{X}_i + F_{bias,i}^{T} \ddot{X}_i + \sum_{i=1}^{N} \frac{1}{2} d_i \ddot{q}_i^{\,2} - \tau_i \ddot{q}_i \right\} \tag{3.2}$$

subject to,

$$A_N^T \ddot{X}_N = b_N \tag{3.3}$$

$$\ddot{X}_{i+1} = {}^{i+1}X_i \ddot{X}_i + \ddot{q}_{i+1} S_{i+1} + \ddot{X}_{bias,i+1} \tag{3.4}$$

This Gauss function ($\mathcal{Z}$) is subjected to linear constraints given by [50],

In the equation 3.2, $\mathcal{Z}$ is the acceleration energy of the kinematic chain, also called as *Zwang* [50]. The function $\mathcal{Z}$ is minimized with respect to joint accelerations, $\ddot{q}$. $\ddot{X}_i$ represents 6 x 1 constrained Cartesian acceleration vector of segment i, expressed in Plücker coordinates. It is given as [25],

$$\ddot{X} = \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \\ \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \tag{3.5}$$

The spatial acceleration vector comprises of linear acceleration (first three elements) and angular accelerations (last three elements).

Further, H_i is the Cartesian space rigid body inertia matrix. $F^T_{bias,i}$ is a vector of bias forces (i.e., external forces, Coriolis and centrifugal forces) acting on segment i.

The equation 3.3 corresponds to Cartesian acceleration constraints in which A_N is a matrix of order 6 x m with m as the number of constraints. The columns of the matrix represents the direction of constraint forces imposed on the end-effector. b_N is a vector of order m x 1 and is called acceleration energy set-point. In addition to Cartesian constraints, the kinematic chain structure is subjected to joint constraints given by equation 3.4. Here, ${}^{i+1}X_i$ is a homogeneous transformation matrix, that transforms Cartesian acceleration vector $(\ddot{X}_{i+1})$ with respect to $\ddot{X}_i$. The term S_{i+1} denotes *motion subspace matrix* that maps from Joint space to Cartesian space. Contrarily, S^T maps from Cartesian to Joint space.

The Vereshchagin solver is domain-specific, since it exploits the kinematic chain structure. Evaluating for a possible minimum solution to the function 3.2, in presence of certain linear constraints is termed as *constrained optimization problem*. The equation is thus extended to [50],

$$\mathcal{Z} = \min_{\ddot{q}} \left\{ \sum_{i=0}^{N} \frac{1}{2} \ddot{X}_i{}^T H_i \ddot{X}_i + F^T_{bias,i} \ddot{X}_i + \sum_{i=1}^{N} \frac{1}{2} d_i \ddot{q}_i{}^2 - \tau_i \ddot{q}_i + \nu_T A_N^T \ddot{X}_N \right\} \tag{3.6}$$

Since the equation 3.2 is subjected to equality constraints (equation 3.3 and 3.4), the quadratic function $\mathcal{Z}$ is minimized by applying the method of Lagrange multipliers [10]. Here, ν is the non-negative *Lagrange multiplier*. In further steps, the solver is derived based on the *Bellman's principle of optimality* [11] [9]. The equation is reformulated as [50],

$$\mathcal{Z}_{i-1}(\ddot{X}_{i-1}, \nu) = \min_{\ddot{q}} \left\{ \frac{1}{2} \ddot{X}^T_{i-1} H_{i-1} \ddot{X}_{i-1} + U_i^T \ddot{X}_i + \frac{1}{2} d_i \ddot{q}_i{}^2 - \tau_i \ddot{q}_i + \mathcal{Z}_i(\ddot{X}_i, \nu) \right\} \tag{3.7}$$

On further solving the equation 3.7 and minimizing with respect to $\ddot{q}$ will yield the solution to a constrained dynamics problem, which is of the form [50],

$$F_N = A_N \nu \tag{3.8}$$

where, F_N is the vector of constraint forces imposed on the segment N.

The outcomes of the optimization problem are computational sweeps that are applied on the kinematic chain to compute true motion at every instance of time. Through these outward and inward sweeps, the solver visits every segments(links) and returns *joint accelerations* ($\ddot{q}$), *Cartesian accelerations* ($\ddot{X}$) and joint torques ($\tau_{control}$) as the solution to the constrained dynamics problem [50].

3.3 Algorithm Description

The algorithm illustrating the computational sweeps in the Vereshchagin solver is described in this section. As specified, the algorithm comprises three recursions - outward, inward and outward. Here, the outward recursion refers to traversing from the fixed base of a kinematic chain to its end-effector. Contrarily, the inward recursion loops from end-effector to base.

The complete algorithm is given below [50] [56] [57],

Algorithm 1: Constrained Hybrid Dynamic Solver

Input : Robot geometry, inertial data, q_i, $\dot{q}_i$, τ_i, $\ddot{X}_0$, F_i^{ext}, A_N, b_N

Output : $\tau_{control}$, $\ddot{q}_i$, $\ddot{X}_i$

begin

/* *Outward sweep of pose, twist and bias components* */

for $i \leftarrow 0$ *to* $N-1$ **do**

$\quad {}^{i+1}_{i}X = \left({}^{d_i}_{p_i}X\,{}^{p_{i+1}}_{d_i}X(q_i)\right)$;

$\quad \dot{X}_{i+1} = {}^{i+1}X_i \dot{X}_i + S_{i+1}\dot{q}_{i+1}$;

$\quad \ddot{X}_{bias,i+1} = \dot{X}_{i+1} \times S_{i+1}\dot{q}_{i+1}$;

$\quad F^b_{bias,i+1} = \dot{X}_{i+1} \times^* H_{i+1}\dot{X}_{i+1} - {}^{i+1}X^*_0 F^{ext}_0$;

$\quad H^A_{i+1} = H_{i+1}$;

$\quad F^A_{bias,i+1} = F^b_{bias,i+1}$;

end

/* *Inward sweep of inertia and force* */

for $i \leftarrow (N-1)$ *to* 0 **do**

$\quad D_{i+1} = d_{i+1} + S^T_{i+1} H^A_{i+1} S_{i+1}$;

$\quad P^A_{i+1} = 1 - H^A_{i+1} S_{i+1} D^{-1}_{i+1} S^T_{i+1}$;

$\quad H^a_{i+1} = P^A_{i+1} H^A_{i+1}$;

$\quad H^A_i = H^A_i + \sum {}^iX^T_{i+1} H^a_{i+1}\, {}^iX_{i+1}$;

$\quad F^a_{bias,i+1} = P^A_{i+1} F^A_{i+1} + H^A_{i+1} S_{i+1} D^{-1}_{i+1} \tau_{i+1} + H^a_{i+1} \ddot{X}_{bias,i+1}$;

$\quad F^A_{bias,i} = F^A_{bias,i} + \sum {}^iX^*_{i+1} F^a_{bias,i+1}$;

$\quad A_i = {}^iX^T_{i+1} P^A_{i+1} A_{i+1}$;

$\quad U_i =$
$\quad\quad U_{i+1} + A^T_{i+1}\left\{\ddot{X}_{bias,i+1} + S_i D^{-1}\left(\tau_{i+1} - S^T_i (F^A_{bias,i+1} + H^a_{i+1}\ddot{X}_{bias,i+1})\right)\right\}$;

$\quad \mathcal{L}_i = \mathcal{L}_{i+1} - A^T_{i+1} S_{i+1} D^{-1}_{i+1} S^T_{i+1} A_{i+1}$

end

/* *Linear constraint force magnitudes* */

$\nu = \mathcal{L}_0^{-1}(b_N - A_0^T \ddot{X}_0 - U_0)$;

/* *Outward sweep of acceleration* */

for $i \leftarrow 0$ *to* $N-1$ **do**

$\quad \ddot{q}_{i+1} = D^{-1}_{i+1}\left\{\tau_{i+1} - S^T_{i+1}\left(F^A_{bias,i+1} + H^A_{i+1}\left({}^{i+1}X_i \ddot{X}_i + \ddot{X}_{bias,i+1}\right) + A_{i+1}\nu\right)\right\}$;

$\quad \ddot{X}_{i+1} = {}^{i+1}X_i \ddot{X}_i + \ddot{q}_{i+1} S_{i+1} + \ddot{X}_{bias,i+1}$;

end

end

The required inputs to the algorithm (1) are listed below;

- *Robot model parameters* - A complete robot model defined by rigid body parameters such as mass, inertia, link lengths of individual segments.

- *Joint positions* defined at current time instance (q_i).
- *Joint velocities* ($\dot{q}_i$)
- *Feed-forward joint torques* (τ_i)
- *Cartesian acceleration* at current instance of time defined at the base ($\ddot{X}_0$).
- *External forces* (F_i^{ext}).
- *Unit constrained forces* applied at the end-effector defined as a matrix (A_N).
- *Acceleration energy set-point* defined at the end-effector (b_N).

In the following subsections, the associated equations are illustrated.

3.3.1 Outward sweep : position, velocity and acceleration recursions

The outward recursion solves the forward kinematics problem. In the algorithm, the first *for loop* iterates from the segment 0 (base) to segment $N-1$ (end-effector). During the recursion, it computes the pose, velocity and acceleration quantities of each segment. Furthermore, it calculates the bias forces and initializes rigid body inertia of the kinematic chain.

In the context of the solver, the computation of desired quantities requires three operations such as [50],

1. *Change in reference point* - Coordinate free aspect that instantly maps position, velocity and acceleration numerically.
2. *Change in coordinate frame* - Coordinate frame transformation to compute entities such as position, velocity and acceleration from proximal joint pose frame $\{p_i\}$ to distal joint pose frame $\{d_i\}$
3. Incorporation of these entities with respect to joint $\{i+1\}$.

The pose from the segment i to $i+1$ is denoted as ${}^{i+1}_{i}X$. This is calculated by the combined transformation between proximal and distal segment frames attached to link (refer to figure 3.3.1). The two transformation matrices are,

- ${}^{d_i}_{p_i}X$ - pose from current (proximal) segment p_i to distal pose frame d_i and;
- ${}^{p_{i+1}}_{d_i}X$ - pose transformation of distal segment d_i to segment p_{i+1}.

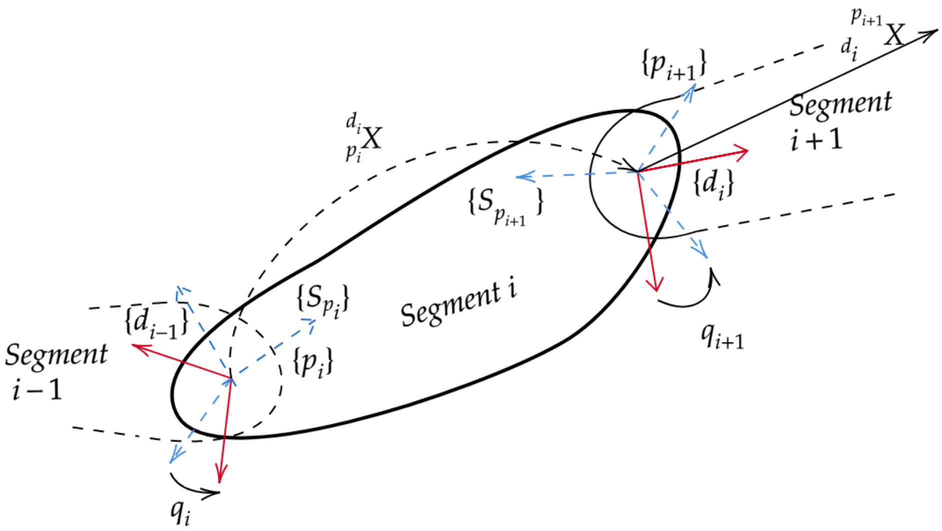

Figure 3.1: Proximal and distal segment frames attachment in a generic kinematic chain and transformation between them [50].

In line 4, the *spatial velocity vector* of segment $i+1$ is calculated, which is represented by $\ddot{X}_{i+1}$. The expression is evaluated as the summation of ${}^{i+1}X_i\ddot{X}_i$ and $S_{i+1}\dot{q}_{i+1}$ recursively. Here the first term represents velocity of segment i expressed in the coordinates of segment $i+1$. The transformation from link i to $i+1$ is computed by matrix ${}^{i+1}X_i$. The second term refers to joint velocity contributions $(\dot{q}_{i+1})$ that is expressed using motion subspace matrix (S_{i+1}).

The next equation (line 5) denotes *bias acceleration* at segment $i+1$, noted as $\ddot{X}_{bias,i+1}$. Since the joint acceleration components are unknown at this stage, only the bias acceleration is computed, provided the Cartesian and joint space acceleration of previous link. Here, $\dot{X} \times S\dot{q}$ acts as time derivative of S, that maps from velocity to acceleration domain.

Furthermore, *bias forces* are determined by the expression in line 6, given the Cartesian velocity vector, $\dot{X}_{i+1}$ and inertia matrix, H_{i+1}. The term $\times^*$ is the cross product operator expressed in Plücker coordinates (refer to appendix section B for explanation on spatial cross products). The bias forces is influenced by the *external forces* as well, given by $F^{ext}_{0,i+1}$ and is transformed from base to end-effector coordinates, expressed by transformation matrix for force vectors, ${}^{i+1}X^*_0$. See the appendix section C for coordinate transformation on force and motion vectors.

In the line 7 and 8, *articulated body inertia* and *articulated bias forces* respectively are initialized with *rigid body* quantities. These values are further used in inward sweep.

3.3.2 Inward sweep : force and inertia recursions

A set of recursive equations in inward sweep computes force and inertial parameters of every segment. The joint torques and external forces acting on the distal segments collectively generates *inertia-dependent acceleration* on the proximal segments [50].

In line 11, the combined inertias of segment $i + 1$ and joint rotor inertia (d_{i+1}) is computed. Matrix P_{i+1} is a projection matrix, that projects *articulated body inertia* and *bias forces* over joint subspace [50] [57]. In further steps, the algorithm calculates *apparent inertia* (line 13) represented as H^a_{i+1}, which is the inertia contributions from the child segments. And *articulated body inertia* (line 14) denoted as H^A_i is calculated by adding all the apparent inertias. Similarly, apparent ($F^a_{bias,i+1}$) and articulated bias forces ($F^A_{bias,i}$) are computed by expression in line 15 and 16 respectively.

In expression 17, *constraint force matrix* is computed (A_i), in which the term $P_{i+1}A_{i+1}$, represents apparent unit constraint forces. Consequently, these constraint forces, external forces and joint torques inclusively generates acceleration energy [50], which is recursively accumulated in vector U_i (line 18). Here, U_i is *desired acceleration energy* vector expressed in Cartesian space. The expression within curly braces denotes acceleration originated from joint torques, and inertial forces applied at distal joints [50].

The inward recursion also deals with constraint acceleration energy, b_N, that is produced by corresponding columns of constraint forces, A_N [50]. This is represented by $\mathcal{L}_i$ (line 19) and is called *constraint coupling matrix*, which is of order $m \times m$ (m is number of constraints). More clearly, each rows in $\mathcal{L}_i$ corresponds to acceleration energy generated by all the constraint forces and accelerations, up until that instance of recursion.

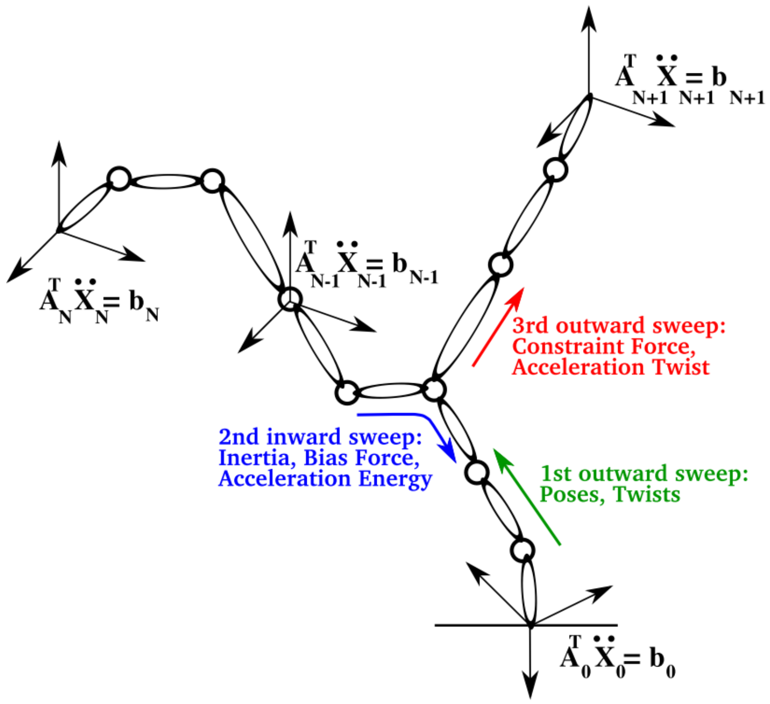

Figure 3.2: An abstract representation of computational sweeps in a kinematic chain, along with computed physical entities and constraints (source: [50])

.

3.3.3 Computing constraint force magnitudes, ν

After reaching the base ($i = 0$), the *constraint force magnitudes* are calculated (line 21). This expression is obtained after minimizing the Equation 3.7 with respect to ν [50]. The constraint force magnitude is a scaling factor that is computed by ratio of generated acceleration energy ($\mathcal{L}_0$) to required acceleration energy, ($b_N - A_0^T \ddot{X}_0 - U_0$). The term $\ddot{X}_0$ denotes the Cartesian acceleration at the base.

Since the base is rigidly fixed in a kinematic chain, $\ddot{X}_0$ is equal to acceleration due to gravity.

It is however important to ensure that the matrix $\mathcal{L}_i$ is of full rank. But this case fails during singularity. To overcome this situation, $(\mathcal{L}^{-1})$ can be computed using the *damped least squares* method, as mentioned in [50].

3.3.4 Outward sweep : Control torques and link accelerations

In the outward sweep, the control torques and joint accelerations of the constrained motion are computed (line 23 and 24) [50]. After minimizing the equation 3.7 with respect to ν in previous step, the *constraint force magnitudes* is substituted and solved for joint acceleration $\ddot{q}_i$ in the final outward sweep. As mentioned before, the joint $i+1$ experiences external and Coriolis forces $(F^A_{bias,i+1})$, inertial forces $(H^A_{i+1}\,{}^{i+1}X_i\ddot{X}_i)$ and feed-forward torques (τ_{i+1}) from the connected child segments. Corresponding to these quantities, the equation in curly braces (line 23) represents the overall control torque that is required to drive the constrained system [57].

Reformulating the expression in line 23 and representing the torque components as (see equation 3.9) [57],

$$\ddot{q}_{i+1} = D^{-1}_{i+1}\{ \overbrace{\tau_{i+1}}^{\text{input torque}} - \underbrace{S^T_{i+1}\big(F^A_{bias,i+1} + H^A_{i+1}\big({}^{i+1}X_i\ddot{X}_i + \ddot{X}_{bias,i+1}\big)\big)}_{\text{bias torque}} - \overbrace{S^T_{i+1}A_{i+1}\nu}^{\text{constraint torque}} \} \tag{3.9}$$

In the final step of the algorithm, spatial acceleration $\ddot{X}$ is computed (line 24) by substituting $\ddot{q}$ from the previous step.

Figure 3.3.2 describes the computational sweeps in a kinematic chain.

3.4 Task Specification

The Popov-Vereshchagin Hybrid Dynamic solver computes the desired motion of manipulator accounting to the task specification. The inputs to the solver includes three kinds of task definitions - *External force*, *Cartesian acceleration constraints* and *feedforward torque*. This section further explains these task definitions.

1. ***External forces*** *(F_{ext}):*
   The external forces (physical or virtual) applied to the end-effector can be used for *impedance control* in Cartesian space [52]. The resulting impedance control is required to ensure a compliant behavior of end-effector [7].

2. ***Cartesian acceleration constraints:***
   The task requirements imposes *Cartesian acceleration constraints* on manipulator motions. There are two distinct types of constraints that can be applied on the segments - *physical* and *virtual*. The former refers to environmental contacts, whereas the latter defines the desired Cartesian accelerations as specified by the user [50].

   Consider a manipulator with N segments. The *virtual* constraints are specified in matrix A_N of order $6 \times m$, where m is number of constraints. The columns of the matrix represent direction of constraint forces being applied on end-effector. As mentioned in section 3.3.2, the acceleration constraints produces *acceleration energy*, represented by b_N. The expression for linear constraints is given in equation 3.3. For instance, if the user defines partial constraints to restrict the motion of a segment in x and z directions linearly, then the A_N matrix can be written as follows,

$$A_N = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.10}$$

   As the motion is restricted, the accelerations should be 0 in x and z directions. The acceleration energy vector can be specified as,

$$b_N = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.11}$$

   Similarly, acceleration constraints can be defined in all six dimensions. Correspondingly, the acceleration energy vector must be specified.

3. ***Feedforward torque** (τ)**:* The input torque is equivalent to joint constraints. This can be used in tasks that require posture control. For instance, in case of manipulators, to remain in a vertical orientation, the joints are provided with feedforward torques [48].

In case of a high level task specification, all the three types of task definitions (as explained previous section) are provided as input to the robot controller. The general control scheme (figure 3.4) including the solver is presented in [57] as,

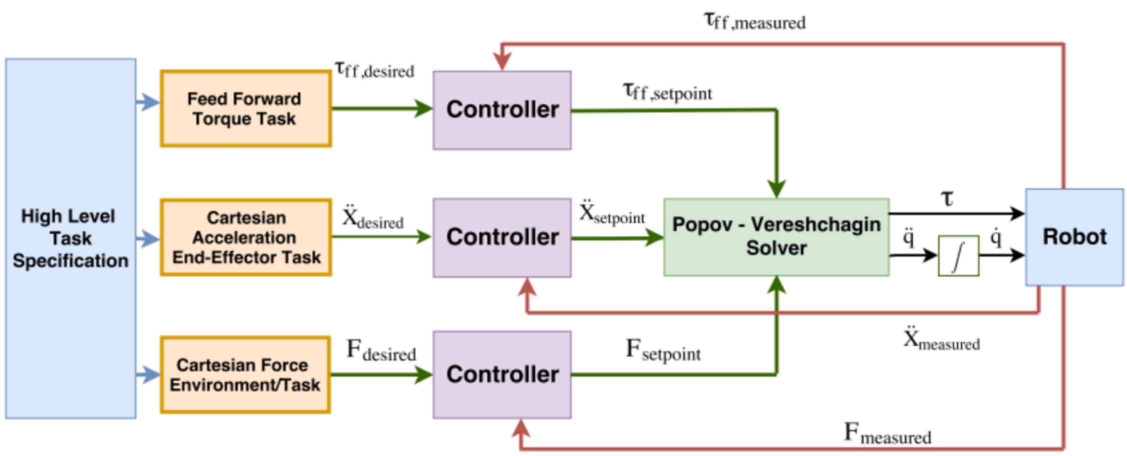

Figure 3.3: General control scheme including the Vereshchagin solver (source: [57])

3.5 Solver Implementation

The Vereshchagin solver is currently implemented using Kinematics and Dynamics library [43], by Herman Bruyninckx, Azamat Shakhimardanov and Ruben Smits. The library is an *open source* and the solver is currently built in C++ programming language.

4

Extending the Vereshchagin hybrid dynamic solver to mobile robots

As discussed in State of the art, there are various software frameworks and dynamic solvers employed specifically in manipulators to satisfy the constraints imposed by task requirements. These approaches considers the dynamic properties of the system and compute the desired motion. In case of mobile robots, there are approaches introduced to compute the control commands by considering the dynamics of the robot. However, there are no task specification frameworks or solvers that would provide a better procedure to handle instantaneous task specifications. Therefore, the main objective of the project is to extend and apply the *Popov-Vereshchagin hybrid dynamic solver* to mobile robots.

The main feature of the solver is that, given the task requirements, it calculates the instantaneous joint accelerations and control torques of a single end-effector. However, this can be extended to compute the desired motion of multiple end-effectors [50]. An autonomous mobile robot can be modeled as kinematic tree structure with wheels as end-effectors. Following sections describe extensions to tree structure and how the extended algorihtm can be applied to mobile robot.

4.1 Extension to kinematic trees

A theoretical description on extension of the solver to multiple end-effectors (kinematic tree structure), is presented by author Azamat Shakhimardanov in his dissertation [50]. The conceptual and algorithmic explanation of this extension in each of the computational sweeps is presented below.

4.1.1 Initial outward sweep

The computations in outward sweep remains unchanged except the way of recursion. For a simple serial chain, the outward sweep is a single path from base to end-effector. This remains the same for kinematic tree structure as well, but the recursion must traverse to all the respective end-effectors [50].

In the Constrained hybrid dynamics algorithm (1), the initial outward sweep loops from $i = 0$ (base) to $N - 1$ (end-effector). To traverse from base to all end-effectors, breadth-first search is used. The tree elements must be ordered in such as way that, iterating through them would result in *breadth-first search*.

4.1.2 Inward sweep

In the current solver, the inward sweep loops from $i = N - 1$ to $i = 0$ using *for loop*. For tree structure, the inward sweep must traverse from all respective end-effectors to base similar to *reverse breadth-first search* fashion.

The main modifications in inward sweep includes [50],

- The apparent inertias (H^a) and forces (F^a) of parent segment must combine all the articulated inertias and forces computed by the child segments.
- The constraint force matrix (A), must combine the constraints applied at each of the end-effectors. Hence, the matrix will be expanded column-wise.
- The acceleration energy (b_N) is accumulated corresponding to the constraints arising from each of the end-effectors.
- At the branching point, the constraints from each of the child segments must be joined into the acceleration constraint coupling matrix ($\mathcal{L}$). The matrix expands block-wise with the increase in number of constraints. Consider

a three different dimensional constraint joined at the branching point, the resulting $\mathcal{L}$ matrix is [50],

$$\mathcal{L} = \begin{pmatrix} \mathcal{L}^k & 0 & 0 \\ 0 & \mathcal{L}^l & 0 \\ 0 & 0 & \mathcal{L}^m \end{pmatrix} \tag{4.1}$$

In the above matrix, k, l and m are dimensions of constraints arising from each of the sub-chains. $\mathcal{L}^k, \mathcal{L}^l$ and $\mathcal{L}^m$ are constraint coupling matrices of respective sub-chains.

4.1.3 Resolving constraints at the base

As explained in the section 3.3.3, the Gauss function, $\mathcal{Z}$ is minimized by applying the method of Lagrange multipliers. This results in constraint force magnitudes ν that corresponds to the acceleration constraints applied at the end-effector. But in case of tree structure, there are multiple end-effectors and corresponding constraint magnitudes must be computed.

citation here........

$$U^A_{desired,0} = U^A_0 - b_N \tag{4.2}$$

$$\nu_{float} = \left[(A^A_0)^T (H^A_0)^{-1} A^A_0 - \mathcal{L}^A_0\right]^{-1} \left[U^A_{desired,0} - (A^A_0)^T (H^A_0)^{-1} F^A_{bias,0}\right] \tag{4.3}$$

The constraint calculation for a fixed base is given in the algorithm 1 (expression in line 21). For a floating base, ν cannot be directly calculated since the base acceleration is unknown. The constraint magnitudes are computed using the expression 4.3. Here, A^A_0 is a extended constraint force matrix for all the end-effector constraints expressed in root coordinates. H^A_0 is *articulated-body inertia* denoted in Plüker coordinates. It is given by [25],

$$H^A_0 = \begin{pmatrix} I_0 & H_0 \\ H^T_0 & M_0 \end{pmatrix} \tag{4.4}$$

The notation $\mathcal{L}_0$ is acceleration coupling matrix expressed as 4.1 for given number of constraints. Further, the desired acceleration energy vector ($U^A_{desired,0}$) expressed at root coordinates is given by 4.2, where U^A_0 is the computed acceleration energy expressed at base coordinates and b_N is the respective constraint acceleration energy vector.

In case of kinematic chain structure, the base is fixed. Hence the root acceleration is equal to acceleration due to gravity (expression 4.5).

$$\ddot{X}_0 = -\ddot{X}_g \tag{4.5}$$

In case of a free floating base (for example: mobile robots, orbiting spacecraft) the base acceleration can be computed by [56].

$$\ddot{X}_{float,0} = -(H^A_0)^{-1}(F^A_{bias,0} + A^A_0 \nu_{float}) \tag{4.6}$$

4.1.4 Final outward sweep

The computations in final outward sweep remains the same, besides the recursion, which must traverse from root to all the end-effectors. The calculated ν_{float} is substituted in expression 23 and joint accelerations ($\ddot{q}$) are computed. Further in line 24, Cartesian acceleration is calculated.

4.2 Conclusion

The *Vereshchagin solver* is exercised in robot manipulators to evaluate hybrid dynamics problem. [43] The algorithm has been implemented using KDL library (open-source), by Ruben Smits, Herman Bruyninckx, and Azamat Shakhimardanov [43]. The KDL provides a framework to model and compute solutions to kinematics and dynamics problem. The real-time code that implements solver on manipulator is provided and maintained under *Orocos (Open Robot Control Software) Project* and is written in C++ programming language. The approach that extend and apply the algorithm to mobile robots is described in the next chapter.

5

Methodology

This chapter describes the applied robot platform used to test the extended solver. To begin with, the robot has to be modeled as a kinematic tree structure. Therefore, the following sections briefly discuss the applied robot specifications and further describes the representation of this robot as a kinematic tree structure.

5.1 Robot Specifications

The applied robot platform is MPO-700 (figure 5.1) [1]. MPO-700 is an omni-directional base employed in high-end service robots [3]. The base features four *Neobotix omnidirectional modules* that enables smooth motion in X-Y directions. When compared to other robot bases with omnidirectional drive kinematics, the MPO-700 has great maneuverability, steadiness, high stability and compact [3]. Hence, it is used in wide range of applications. One of the popular platform built based of MPO-700 is *Care-O-bot* 3 developed by Fraunhofer IPA[1].

The MPO-700 base has four Castor wheels and four drive wheels. The drive wheels are at an offset from castor wheels. In the default configuration, the drive wheels are oriented inwards by 45^0 (as seen in figure 5.1). A controlled motion can only be acheieved if all the drives coordinated properly. The robot base is also equipped with two SICK sensors (laser scanners), whose data is used for obstacle avoidance. There are two emergency stop buttons placed on either side of the base.

[1]Fraunhofer IPA - http://www.ipa.fraunhofer.de

In case of danger, the robot motion can be stopped immediately by pressing these emergency buttons. On the base, there is a LC-Display, which displays the detailed information on current state of the robot. A key witch is present slightly above laser scanner. This is used to start or stop the robot [1].

Figure 5.1: MPO-700 Neobotix (source: [2])

5.2 Kinematic Tree Representation

A kinematic tree comprises of interconnected segments (links). A typical kinematic tree description is provided by KDL library. Representation of a single segment (KDL::Segment) is given in figure 5.2 [5].

A KDL segment (figure:5.2) is composed of four frames,

1. $F_{reference}$: A *reference* frame (black colored frame) with respect to which other frames are expressed.

2. F_{joint}: A one DOF *joint* frame (red) expressed about joint axis. The orientation of the frame is same as $\{F_{reference}\}$ and translation is given by $\{p_{origin,a}\}$. The frame is defined in KDL::Joint class.

3. $F_{inertia}$: A *Cartesian space inertia matrix* (green) expressed with respect to *tip frame* and $p_{inertia,a}$ is the translation vector. The frame is defined in KDL::RigidBodyInertia class.

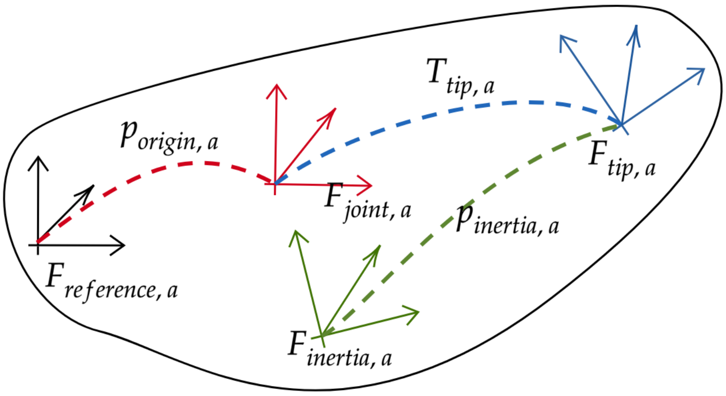

Figure 5.2: KDL Segment (source: [5])

4. F_{tip}: Frame attached at the tip of a segment. As seen in the figure 5.2, $\{F_{tip,a}\}$ is defined with respect to joint frame (blue) and transformation is given by $\{T_{tip,a}\}$ (by default: $\{T_{tip,a}\}$ is identity transformation).

A Kinematic tree is simply a composition of these KDL segments. An example is shown in the below figure (5.3) which describes a *Kinematic tree* with two branches [5]. Here, *Segment a* is the *root* of the tree and *Segment b* and *Segment c* are *child branches.* According to the convention, the joint frame of the succeeding segment is attached to tip frame of the preceding segment. Therefore, the tip frames acts as the reference frame for the succeeding segments ($F_{tip,a} = F_{reference,b} = F_{reference,c}$). Similarly, the representation can be extended for multiple chains and interconnected segments. Referring to this representation, a tree structure for the MPO-700 base is created in next section.

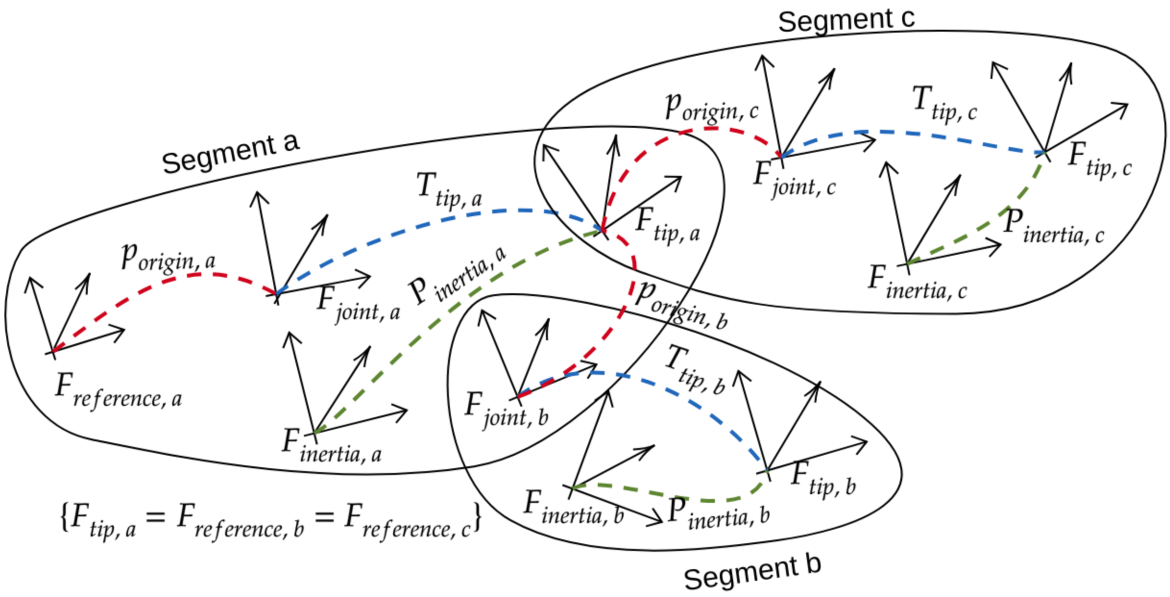

Figure 5.3: Kinematic tree representation in KDL (source: [5])

5.2.1 MPO-700 base as kinematic tree

The MPO-700 robot base is modeled as kinematic tree by defining the robot chassis as *root* of the tree. As seen in the figure 5.4, the *root frame* is defined at the center, on the surface of chassis. Using the technical dimensions obtained from the operating manual and URDF model of base, further tree elements are designed [1].

A sub-chain is defined from root to the drive wheel's point of contact on the ground. Each of these sub-chains comprises of three segments. However, only the *tip frame* of the segments are shown in the figure. One of the sub-chains is explained below.

- B: represents link from root frame to the frame (green) at the corner of the chassis. Physically this link denotes translation from tip of root segment to tip of segment B2. Here, the z_{b2} axis coincides with the castor wheel's axis.
- C: denotes a link from corner of the chassis (z_{b2}) to the center of the drive wheel (z_{c2}). Please note that the drive wheel is at an offset from castor and is oriented by 45^0. Hence, C2 represents transformation between these two

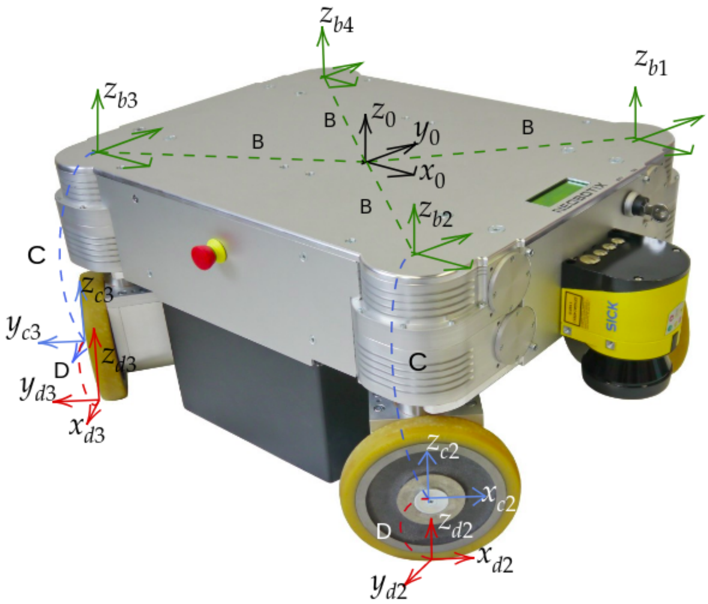

Figure 5.4: Representation of tree structure on MPO-700 [source: [2]](Note: The links with same name represents identical transformation)

frames.

- D: defines the link from center of drive wheel to its point of contact on the ground. Physically, D represents the wheel radius.

The KDL::Tree class adds all the sub-chains to the root segment. A 2D view of this tree structure is shown in the figure 5.5. As previously mentioned, the outward and inward recursions are achieved similar to *breadth-first search* and *reverse breadth-first search* respectively. The tree elements must be ordered in such a way that, the recursion imitates these search patterns. When the tree structure is created, the KDL::SegmentMap function arranges the tree elements lexicographically. Hence, the segment names follow lexicographical order (as seen in figure 5.5).

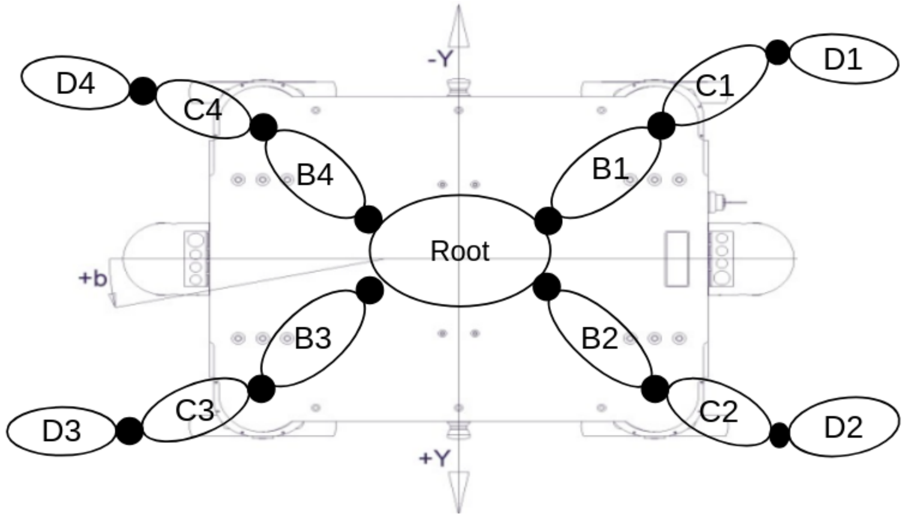

Figure 5.5: Kinematic tree structure of MPO-700 robot base (source: [1])

5.3 Implementation details

Currently the Vereshchagin solver for kinematic chains is implemented in KDL library [43] using C++ programming language. The available code base for the current solver is developed and maintained by the Ruben Smits, Herman Bruyninckx, Azamat Shakhimardanov [43]. In the chapter 4, the extended algorithm for kinematic tree was proposed. Further interest lies in implementing this proposed algorithm on MPO-700 robot base. As described above, the robot base is modeled as tree structure in KDL library. Further, the existing code base is extended to implement the proposed algorithm.

6

Experimental Evaluation and Results

The following chapter presents the *proof of concept* and evaluation of the proposed extension to *Popov-Vereshchagin solver* . All the experiments are conducted in a simulation environment. The obtained results are analyzed based on the physical behavior of the system.

6.1 Experimental Setup

This section describes the overall experimental setup followed to show the proposed extension to the solver for a mobile base. All the experiments discussed in the following sections are tested in a simulation environment. There are mainly three cases discussed in experimental setup. By varying the inputs to the solver, the obtained simulation results are analyzed based on the possible physical behavior of the system.

The inputs to the solver are, robot model, inertia data, q, $\dot{q}$, τ, $\ddot{X}_0$, F^{ext}, A_N and b_N. The robot model is the created kinematic tree structure of the MPO-700 robot base, in the previous chapter. Some of the input parameters that are kept constant throughout the experiment, they are,

- *Input joint angles (q), velocities ($\dot{q}$) and accelerations ($\ddot{q}$)* $= 0$ for all joints
- *Linear constraint matrix (A_N):* This defines the directions of the acceleration constraints on the end-effectors (wheels). In our case, the wheels are

constrained along *linear-y* (sliding constraint), *linear-z* (no acceleration perpendicular to the ground) and *angular-x* (wheel should not "roll" about x-axis). The A_N matrix described for all the wheels is given by,

$$A_N = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{6.1}$$

- *Beta vector (b_N):* This defines the acceleration energy vector corresponding to directions of applied constraints. Since the wheels are constrained to not to have acceleration along linear-y, linear-z and angular-z, the b_N vector for each wheels is given by,

$$b_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{6.2}$$

The two other input parameters are external force and feed-forward torques. In the following experiment scenarios, these two parameters are varied and the results are analyzed. The solver outputs are $\ddot{q}$, $\tau_{control}$ and $\ddot{X}$.

- *Joint accelerations ($\ddot{q}$):* Represents the resultant accelerations at every joints. For simplified notations, the orientation joints are denoted by "o" and drive joints are denoted by "d". In the table below, the order of all the joint acceleration are - [o1, d1, o2, d2, o3, d3, o4, d4].

- *Base acceleration:* Describes the Cartesian acceleration of the mobile base.

- *Control torques ($\tau_{control}$):* Represents the resultant control torques of every joints (o1, d1, o2, d2, o3, d3, o4, d4).

Below table displays the simulation results. In the following sections each of the cases and its results are analyzed.

Inputs		Outputs		
External force (F_{ext})	Feedforward Torques(τ)	Joint accelerations ($\ddot{q}$)	Base acceleration	Control torques ($\tau_{control}$)
$\tau_z = 10$ $[0, 0, 0, 0, 0, 10]$	$[0, 0, 0, 0, 0, 0, 0, 0]$	$[-0.651743, -1.36504, -3.97664, -1.36552,$ $-3.97664, -1.36552, -0.651743, -1.36504]$	$\dot{\omega}_z = 1.4962$ $\dot{v}_x = -0.0368095$	$[0.303226, -0, -0.126057, -0,$ $-0.126057, -0, 0.303226, 0]$
$f_x = 50$ $[50, 0, 0, 0, 0, 0]$	$[0, 0, 0, 0, 0, 0, 0, 0]$	$[3.49264, 0.168407, 3.901640.168467,$ $3.90164, 0.168467, 3.49264, 0.168407]$	$\dot{\omega}_z = -0.184047$ $\dot{v}_x = 0.158089$	$[0.243069, -0, 0.238214, -0,$ $0.238214, -0, 0.243069, -0]$
$F_{ext} = [0, 0, 0, 0, 0, 0]$	$\tau_{d_1} = 1.0$ $\tau_{d_2} = 1.0$ $\tau_{d_3} = 1.0$ $\tau_{d_4} = 1.0$	$[0.329141, 34.0727, 0.549795, 34.0727,$ $0.757357, 34.1083, 0.12158, 34.1082]$	$\dot{v}_x = 0.0111047$ $\dot{v}_y = 0.00467013$ $\dot{\omega}_z = -0.192697$	$[0.0695287, -0, -0.0665266, -0,$ $-0.0539123, -0, 0.0463929, -0]$

Table 6.1: Experimental analysis

6.2 Experiment 1:

The first experiment corresponds to the first row in table 6.1. As mentioned above, the quantities, q, $\dot{q}$, $\ddot{q}$, A_N and b_N are kept constant. In the first experiment, only the external torque (τ_z) of 10 units is applied to the base and joint torques is zero. On applying torque of 10N on a rigid-body system, it is expected to produce certain angular acceleration at base. The relation between torque and angular acceleration is given by,

$$\tau = I_z \dot{\omega}_z \tag{6.3}$$

Here, I_z represents the rotational inertia of the body. For the base, $I_z = 3.69$ as obtained from the URDF model. From the above relation, the angular acceleration will be $\dot{\omega}_z = 2.71$ rad$/s^2$. This result is

Justifications on joint acceleration and control torques

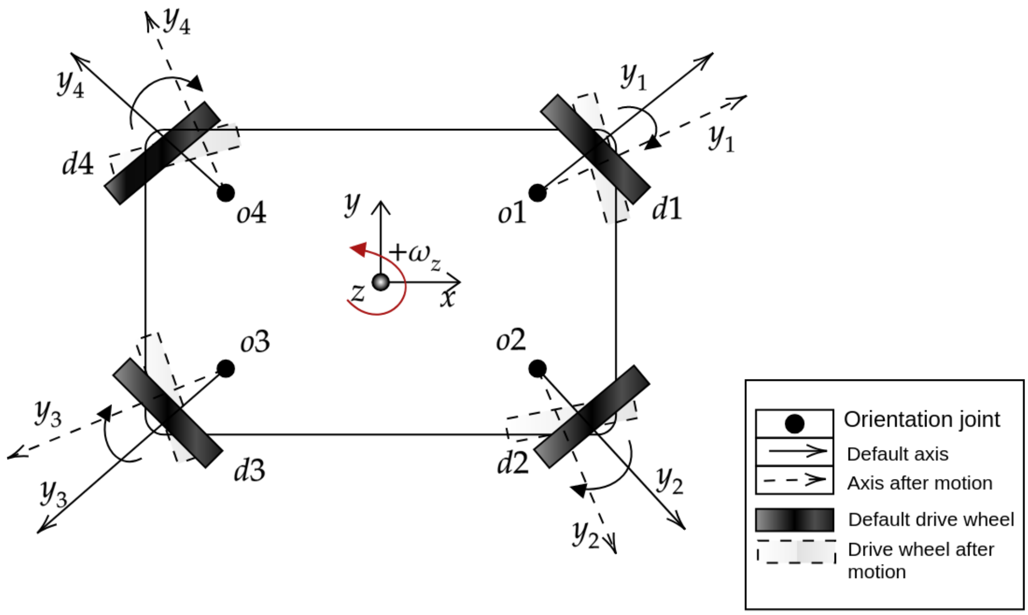

Figure 6.1: Experiment 1

6.3 Experiment 2

In this experiment, an external force in linear x direction is applied on the robot base, i.e., $f_x = 50$ and joint torques is 0. This external force causes the castor wheels to align in the direction of applied force and results in a non-zero linear acceleration at base ($\dot{v}_x = 0.158$). The well-known relation between force and acceleration according to Newton's second law of motion [40] is,

$$f_x = m\dot{v}_x \tag{6.4}$$

Here, $m = 180$, mass of base (as given in the URDF model of MPO-700). By substituting the known variables in the equation 6.4, yields linear acceleration $\dot{v}_x = 0.277$ SI units. The error (δ) between the calculated and the obtained acceleration value, $\delta = 0.119$. (Justification on error.......)

Additionally, the applied force produces angular acceleration about z. The relation between linear force and angular acceleration is,

....

6.4 Experiment 3

In this experiment, no external force is given, the joint torques are applied to drive wheels that are equal to 1.0 SI units. In a physical system, the joint torques applied to the wheels result in acceleration of the system. The results show that the base has acceleration along linear-x, linear-y and angular-z.

7

Conclusions

7.1 Contributions

This project presents an extension to a *Constrained hybrid dynamic algorithm* called *Popov-Vereshchagin solver*. The existing approach is implemented for kinematic chain structure (robot manipulators). The main feature of the solver is

7.2 Lessons learned

7.3 Future work

A

Dynamic equation of motion

The general dynamic equation of motion of a rigid body is expressed as [25] [50],

$$M(q)\ddot{q} + C(q,\dot{q}) = \tau \tag{A.1}$$

where, $M(q)$ represents mapping from motion domain(M^n) to force domain(F^n). $C(q,\dot{q})$ is the Coriolis and Centrifugal forces acting on the rigid body. Both these quantities are dependent on q, $\dot{q}$, $\ddot{q}$ and the physical model of rigid body [25]

The dynamics problem is divided into forward and inverse dynamics. Computing the acceleration($\ddot{q}$), given the input forces(τ) is termed as *forward dynamics* problem. Conversely, *Inverse dynamics* problem calculates the forces, τ given accelerations $\ddot{q}$.

The rigid body is generally subjected to various motion constraints that changes the form of the dynamics equation. The extended equation is given by [50],

$$M(q)\ddot{q} = \tau_a(q) - \tau_c(q) - C(q,\dot{q}) \tag{A.2}$$

In the above equation, τ_a represents input forces and τ_c is the constraint forces from the task specification.

B

Plücker Notations for Spatial cross products

There are two spatial cross product operators expressed using Plücker notations are, $\times$ and $\times^*$ [25]. The operators can be regarded as dual to each other. The matrix representation of these operators are deduced as [25],

$$\hat{v}_O \times = \begin{bmatrix} \omega \\ v_O \end{bmatrix} \times = \begin{bmatrix} \omega\times & 0 \\ v_O\times & \omega\times \end{bmatrix} \tag{B.1}$$

and,

$$\hat{v}_O \times^* = \begin{bmatrix} \omega \\ v_O \end{bmatrix} \times^* = \begin{bmatrix} \omega\times & v_O\times \\ 0 & \omega\times \end{bmatrix} \tag{B.2}$$

C

Representation of coordinate transforms

In this section, the notations used to represent coordinate transformation matrices on motion and force vectors is presented. This follows the convention given in [25].

- ${}^{i+1}X_i$ - denotes coordinate transform from i to $i+1$ coordinates of a motion vector,
- ${}^{i+1}X_i^*$ - denotes coordinate transform from i to $i+1$ coordinates of a force vector.

These transforms are related by the following equation [25],

$$ {}^{i+1}X_i^* = {}^{i+1}X_i^{-T} $$

References

[1] MPO-700-Operating Manual, . URL www.neobotix-roboter.de. Version 2.4.3, edited 23. April 2018 in Heilbronn, Germany.

[2] Neobotix GmbH, MPO-700-Datenblatt, . Weipertstr. 8-10, 74076 Heilbronn, Germany. Available: https://www.neobotix.de.

[3] *Omni-directional robot MPO-700*. URL https://www.neobotix-robots.com/omnidirectional-robot-mpo-700.html. [Online] Last Accessed: 2019-01-02.

[4] The *iTaSC* software. URL http://www.orocos.org/wiki/orocos/itasc-wiki/2-itasc-software. [Online] Last Accessed: 2019-01-03.

[5] *Kinematic Trees:The Orocos Project*. URL http://www.orocos.org/wiki/main-page/kdl-wiki/user-manual/kinematic-trees. [Online] Last Accessed: 2018-12-31.

[6] Lounis Adouane, Ahmed Benzerrouk, and Philippe Martinet. Mobile robot navigation in cluttered environment using reactive elliptic trajectories. In *18th IFAC World Congress*, 2011.

[7] Alin Albu-Schaffer and Gerd Hirzinger. Cartesian impedance control techniques for torque controlled light-weight robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 657–663. IEEE, 2002.

[8] JR Asensio and L Montano. A kinematic and dynamic model-based motion controller for mobile robots. *IFAC Proceedings Volumes*, 35(1):427–432, 2002.

[9] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[10] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods.* Academic press, 2014.

[11] Dimitri P Bertsekas et al. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, Massachusetts, 1996.

[12] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on systems, Man, and Cybernetics*, 19(5): 1179–1187, 1989.

[13] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots in cluttered environments. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 572–577. IEEE, 1990.

[14] Herman Bruyninckx and Joris De Schutter. Specification of force-controlled actions in the" task frame formalism"-a synthesis. *IEEE transactions on robotics and automation*, 12(4):581–589, 1996.

[15] Herman Bruyninckx and Oussama Khatib. Gauss' principle and the dynamics of redundant and constrained manipulators. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 3, pages 2563–2568. IEEE, 2000.

[16] Herman Bruyninckx et al. Kinematic models for robot compliant motion with identification of uncertainties. *PhD thesis, KU Leuven, Department of Mechanical Engineering*, 1995.

[17] Kyong-Sok Chang and Oussama Khatib. Operational space dynamics: Efficient algorithms for modeling and control of branching mechanisms. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 850–856. IEEE, 2000.

[18] François Chaumette and Éric Marchand. A redundancy-based iterative approach for avoiding joint limits: Application to visual servoing. *IEEE Transactions on Robotics and Automation*, 17(5):719–730, 2001.

[19] Joris De Schutter and Hendrik Van Brussel. Compliant robot motion i. a formalism for specifying compliant motion tasks. *The International Journal of Robotics Research*, 7(4):3–17, 1988.

[20] Joris De Schutter, Tinne De Laet, Johan Rutgeerts, Wilm Decré, Ruben Smits, Erwin Aertbeliën, Kasper Claes, and Herman Bruyninckx. Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *The International Journal of Robotics Research*, 26 (5):433–455, 2007.

[21] Wilm Decré, Ruben Smits, Herman Bruyninckx, and Joris De Schutter. Extending itasc to support inequality constraints and non-instantaneous task specification. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 964–971. IEEE, 2009.

[22] Wilm Decré, Herman Bruyninckx, and Joris De Schutter. Extending the itasc constraint-based robot task specification framework to time-independent trajectories and user-configurable task horizons. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1941–1948. IEEE, 2013.

[23] A F. Vereshchagin. Computer simulation of the dynamics of complicated mechan-sisms of robot manipulators. *Eng Cybern*, 12, 1974.

[24] Roy Featherstone. Robot dynamics algorithms. 1984.

[25] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[26] Roy Featherstone and David Orin. Robot dynamics: equations and algorithms. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 826–834. IEEE, 2000.

[27] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1): 23–33, 1997.

[28] Shuzhi Sam Ge and Yun J Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots*, 13(3):207–222, 2002.

[29] Dingqiang Han, Xingguang Duan, Meng Li, Tengfei Cui, Anji Ma, and Xiaodong Ma. Interaction control for manipulator with compliant end-effector based on hybrid position-force control. In *Mechatronics and Automation (ICMA), 2017 IEEE International Conference on*, pages 863–868. IEEE, 2017.

[30] Miguel Juliá, Arturo Gil, Luis Payá, and Oscar Reinoso. Local minima detection in potential field based cooperative multirobot exploration. *International Journal of Factory Automation, Robotics and Soft Computing*, 3, 2008.

[31] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[32] Oussama Khatib and Joel Burdick. Optimization of dynamics in manipulator design: The operational space formulation. *INT. J. ROBOTICS AUTOM.*, 2 (2):90–98, 1987.

[33] Oussama Khatib, Luis Sentis, Jaeheung Park, and James Warren. Whole-body dynamic behavior and control of human-like robots. *International Journal of Humanoid Robotics*, 1(01):29–43, 2004.

[34] Oussama Khatib, Luis Sentis, and Jae-Heung Park. A unified framework for whole-body humanoid robot control with multiple constraints and contacts. In *European Robotics Symposium 2008*, pages 303–312. Springer, 2008.

[35] Torsten Kröger, Bernd Finkemeyer, Ulrike Thomas, and Friedrich M Wahl. Compliant motion programming: The task frame formalism revisited. *Mechatronics & Robotics, Aachen, Germany*, 2004.

[36] Nicolas Mansard, Oussama Khatib, and Abderrahmane Kheddar. A unified approach to integrate unilateral constraints in the stack of tasks. *IEEE Transactions on Robotics*, 25(3):670–685, 2009.

[37] Nicolas Mansard, Olivier Stasse, Paul Evrard, and Abderrahmane Kheddar. A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–6. IEEE, 2009.

[38] Eric Marchand and Gregory D Hager. Dynamic sensor planning in visual servoing. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 3, pages 1988–1993. IEEE, 1998.

[39] Matthew T Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(6): 418–432, 1981.

[40] Isaac Newton. *Philosophiae naturalis principia mathematica*, volume 1. G. Brookman, 1833.

[41] Vincent Padois, J-Y Fourquet, and Pascale Chiron. Kinematic and dynamic model-based control of wheeled mobile manipulators: A unified framework for reactive approaches. *Robotica*, 25(2):157–173, 2007.

[42] Min Gyu Park and Min Cheol Lee. A new technique to escape local minimum in artificial potential field based path planning. *KSME international journal*, 17(12):1876–1885, 2003.

[43] H. Bruyninckx R. Smits, E. Aertbelien and A. Shakhimardanov. *Kinematics and Dynamics (KDL)*. URL http://www.orocos.org/kdl. [Online] Last Accessed: 2018-12-30.

[44] Oscar E Ramos, Layale Saab, Sovannara Hak, and Nicolas Mansard. Dynamic motion capture and edition using a stack of tasks. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 224–230. IEEE, 2011.

[45] Layale Saab, O Ramos, Nicolas Mansard, Philippe Souères, and Jean-Yves Fourquet. Generic dynamic motion generation with multiple unilateral constraints. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4127–4133. IEEE, 2011.

[46] Layale Saab, Oscar E Ramos, François Keith, Nicolas Mansard, Philippe Soueres, and Jean-Yves Fourquet. Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *IEEE Transactions on Robotics*, 29(2):346–362, 2013.

[47] J De Schutter and H Van Brussel. Compliant Robot Motion II. A Control Approach Based on External Control Loops. *The International Journal of Robotics Research*, 7(4):18–33, 1988. doi: 10.1177/027836498800700402. URL https://doi.org/10.1177/027836498800700402.

[48] Luis Sentis and Oussama Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(04):505–518, 2005.

[49] Luis Sentis and Oussama Khatib. A whole-body control framework for humanoids operating in human environments. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2641–2648. IEEE, 2006.

[50] Azamat Shakhimardanov. Composable robot motion stack: Implementing constrained hybrid dynamics using semantic models of kinematic chains. 2015.

[51] Dongjun Shin, Irene Sardellitti, and Oussama Khatib. A hybrid actuation approach for human-friendly robot design. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1747–1752. IEEE, 2008.

[52] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[53] Ruben Smits, Tinne De Laet, Kasper Claes, Herman Bruyninckx, and Joris De Schutter. itasc: A tool for multi-sensor integration in robot manipulation. In *Multisensor Fusion and Integration for Intelligent Systems*, pages 235–254. Springer, 2009.

[54] Nikhil Somani, Markus Rickert, Andre Gaschler, Caixia Cai, Alexander Perzylo, and Alois Knoll. Task level robot programming using prioritized non-linear inequality constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[55] Dominick Vanthienen, Tinne De Laet, Wilm Decré, Ruben Smits, Markus Klotzbücher, Koen Buys, Steven Bellens, Luca Gherardi, Herman Bruyninckx,

and Joris De Schutter. itasc as a unified framework for task specification, control, and coordination, demonstrated on the pr2. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011).*, 2011.

[56] AF Vereshchagin. Modeling and control of motion of manipulational robots. *SOVIET JOURNAL OF COMPUTER AND SYSTEMS SCIENCES*, 27(5): 29–38, 1989.

[57] Djordje Vukcevic. Extending a constrained hybrid dynamics solver for energy-optimal robot motions in the presence of static friction. *Technical Report/Hochschule Bonn-Rhein-Sieg-University of Applied Sciences, Department of Computer Science*, 2018.

[58] Michael W Walker and David E Orin. Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, 104(3):205–211, 1982.