Collision avoidance for collaborative robotics

Mohammad Safeea

Supervisor: Dr. Pedro Neto

Thesis
Submitted in
Partial Fulfilment of the Requirements for the
Degree of Master in Mechanical Engineering
Speciality of Production and Project

2nd March 2016

Contents

1 INTRODUCTION **9**
- 1.1 Collision avoidance . . . 9
- 1.2 Tasks and objectives . . . 11
- 1.3 Contribution . . . 12
- 1.4 Nomenclature . . . 12

2 DYNAMICS OF SERIALLY LINKED ROBOTS **14**
- 2.1 Abstract . . . 14
- 2.2 Introduction . . . 14
- 2.3 Theory and principals . . . 18
  - 2.3.1 Link's acceleration due to the single-frame rotation . . . 19
  - 2.3.2 Link's inertial moment due to single-frame effect . . . 20
  - 2.3.3 Dynamical representation of a single-link . . . 22
  - 2.3.4 Moment acting on a joint due to robot dynamics . . . 23
  - 2.3.5 Joint space inertia, Coriolis & Centrifugal matrices . . . 24
  - 2.3.6 Calculating the time derivative of inertia matrix . . . 25
- 2.4 JSIM for hyper-joint manipulators . . . 26
- 2.5 Algorithm and variables organization in computer's memory . . . 29
- 2.6 Implementation and results . . . 29
- 2.7 Appendix I . . . 33
- 2.8 Appendix II . . . 37

3 COLLISION AVOIDANCE **41**
- 3.1 State of the art . . . 41
  - 3.1.1 Potential fields . . . 41
  - 3.1.2 Optimization techniques . . . 42
  - 3.1.3 Heuristic methods . . . 44
  - 3.1.4 Others . . . 44
- 3.2 Proposed approach . . . 45
- 3.3 Implementation overview . . . 47
- 3.4 Potential field method . . . 48
- 3.5 Geometrical representation . . . 49
  - 3.5.1 Representing objects geometry . . . 49
  - 3.5.2 Obstacles representation . . . 49
  - 3.5.3 Robot representation . . . 49

3.6 Minimum distance calculation . . . 50
3.6.1 Minimum distance between two spheres . . . 50
3.6.2 Minimum distance between a cylinder and a sphere . . . 50
3.6.2.1 First configuration . . . 51
3.6.2.2 Second configuration . . . 51
3.6.2.3 Third configuration . . . 52
3.6.3 Minimum distance between two cylinders . . . 52
3.6.3.1 Common normal on two line segments . . . 53
3.6.3.2 Quadratic optimization . . . 55
3.6.3.3 Conclusion: . . . 59
3.7 Mathematical model of the robot . . . 60
3.8 Vectors influencing robot's motion . . . 62
3.8.1 Repulsion vector . . . 62
3.8.2 Attraction vector . . . 64
3.9 Collision avoidance controllers . . . 64
3.9.1 Kinematics control . . . 64
3.9.2 Force control . . . 67
3.9.2.1 Joint space control . . . 67
3.9.2.2 Operational space force control . . . 68
3.9.2.3 Linearising force control equation . . . 70
3.10 Other considerations . . . 70
3.10.1 Workspace limit . . . 71
3.10.2 Joint limits . . . 71
3.10.3 Joint limits in the linearised dynamics approach . . . 72
3.10.4 Self-collision avoidance . . . 74
3.11 Modifying attraction force to reduce the risk of collision . . . 74
3.12 Drawbacks associated with artificial potential field . . . 76
3.12.1 Oscillations . . . 76
3.12.2 Goal Non-Reachable Obstacles Nearby (GNRON) . . . 76

4 Experiments and Results **78**
4.1 Testing set and Results . . . 78
4.1.1 Test 1 . . . 79
4.1.2 Test 2 . . . 83
4.1.3 Test 3 . . . 85
4.1.4 Test 4 . . . 87
4.1.5 Test 5 . . . 89
4.1.6 Test 6 . . . 92
4.1.7 Test 7 . . . 94
4.1.7.1 Simulation 1 . . . 94
4.1.7.2 Simulation 2 . . . 95
4.1.7.3 Simulation 3 . . . 96
4.2 Discussion . . . 98

5 Conclusion and Future work **99**
5.1 Conclusion 99
5.2 Future work 101

List of Figures

1.1 Baxter robot with coworker in a factory, collision avoidance is of vital importance to guarantee the safety of the coworker. 10
1.2 A robot locate in potentially explosive environment, among pipes carrying gas [1]. 10
1.3 Morphin map of Curiosity Mars rover, courtesy of Mars autonomy project. 10

2.1 Inertial moment $\boldsymbol{\mu}_{Cij}$ and linear acceleration $\ddot{\boldsymbol{p}}_{Cij}$ of centre of mass of link i transferred by frame j 18
2.2 Tangential acceleration of centre of mass of link i transferred by frame j 19
2.3 Normal acceleration of centre of mass of link i transferred by frame j . . 20
2.4 Coriolis acceleration of centre of mass of link i transferred from frame j 21
2.5 Dynamical representation of each link. 22
2.6 Moment acting on joint j due to robot's motion 24
2.7 Inertial forces and moments acting on link i due to angular acceleration of joint j. 27
2.8 Representation of links dynamics and joints moments in memory 30
2.9 Number-of-operations with DOF, required for calculating JSIM using CRBA, GDA and GDAHJ algorithms. 32
2.10 Coordinate vectors of point of interest $\mathbf{P}$ on link i of a serial robot. . . . 38

3.1 Collision avoidance between two robots [2] 43
3.2 Collision avoidance between a spherical obstacle and a manipulator [2] . 43
3.3 The main building blocks for collision avoidance 47
3.4 Artificial potential field concept. 48
3.5 A robot represented by cylindrical primitives superimposed on it 50
3.6 Minimum distance between two spheres 51
3.7 Minimum distance between line segment (associated to a cylinder) and a point (center of a sphere) 52
3.8 Minimum distance between two line segments 54
3.9 Region of feasible solutions for the quadratic optimization problem. The origin is outside the transformed region. 57
3.10 Region of feasible solutions for the quadratic optimization problem. The origin is inside the transformed region. 58
3.11 The original optimization problem defined by level sets of objective function and region of feasible solutions, before applying the transform. . . . 60

3.12 Equivalent optimization problem defined by level sets of equivalent objective function and region of feasible solutions, after the transform. . . . 60
3.13 Repulsion vector aligned with minimum distance segment 62
3.14 Magnitude of normalized repulsion vector as function of normalized distance using different functions. 63
3.15 The path curve, the attraction pole, and the error vector 65
3.16 Outline of collision avoidance strategies 65
3.17 Work space limits approximated as spheres in operational space 71
3.18 Drawback of repulsion torques method for incorporating joints limits . . 73
3.19 A curve representing the scaling factor a_i 73
3.20 Modified of attraction force as a function of normalized distance 75
3.21 Goals non-reachable with obstacle nearby 77

4.1 Layout of simulation scene for Test 1 79
4.2 Test 1 results. 80
4.3 Minimum distance between robot and cowrker for Test 1. 81
4.4 End-effector velocity for Test 1. 82
4.5 Test 2 results. 83
4.6 Minimum distance between coworker and robot. 84
4.7 End-effector's velocity. 84
4.8 Test 5 simulation scene. 85
4.9 Test 3 results. 86
4.10 Test 3 minimum distance between robot and coworker. 86
4.11 Test4 results. 87
4.12 End-effector's velocity. 88
4.13 State machine for robotic cell control with collision avoidance capability. 90
4.14 Motion trajectories. 91
4.15 Robotic cell simulation. 91
4.16 Goal non-reachable with obstacle nearby. 92
4.17 Test 6 velocity results. 92
4.18 Test 6 results. 93
4.19 Scene overview. 94
4.20 Third joint angle for simulation 1. 95
4.21 Third joint angle for simulation 2. 95
4.22 End-effector X component of velocity. 96
4.23 Third joint angle, scaling factor method. 97
4.24 End-effector X component of velocity. 97

List of Tables

2.1 Execution time comparison 31
2.2 Operation count 34
2.3 Acceleration terms e_i 38

4.1 Tests parameters (K-kinematics controller, F-force controller). 78

ACKNOWLEDGEMENT

I would like to thank Professor Pedro Neto for his kindness, important notes, and the useful guidance that he kindly offered throughout this work. He provided me with encouragement, and unwavering enthusiasm with my work, his advice and leadership were unequivocally fundamental in the progress of this work.

The most special thanks to President Jorge Sampaio, and his adviser the kind Dr. Helena Barroco, and also to Dr. Teresa Baptista, first for their kindness, second for believing in me, and third for offering me this exquisite opportunity to continue my master studies, without their assistance and continuous support, this work would have never been materialized.

I also express my appreciation to the administration of Coimbra University, for offering the suitable and rich environment to work in, they provided the required resources that facilitated my work, especially the 24/7 unrestricted access to the robotics lab, with all of its resources, and materials, among others.

A special sense of gratitude to my loving family, particularly my mother, for her faith in me, encouragement and moral support, which provided the needed motivation during times of difficulty.

Most special thanks to my friends and colleagues.

Abstract

In this thesis we visit the problem of real-time collision avoidance for robotic manipulators in unstructured and dynamic environments. The main objective will be to implement a human-robot collision avoidance algorithm for a robotic cell that utilizes an industrial robotic manipulator where the human coworker and the robot share the same working area. For this purpose the pioneering work of Khatib in the artificial potential field method was taken as the basis to our work. Thus, an implementation of two different collision avoidance controllers is addressed. The first of which is based on kinematics, while the other is based on force control that take into consideration robot's dynamics.

For developing the force controller, and for achieving real time performance of the virtual-reality simulations, we had to implement a light weight numerical method for computing robot's dynamics. Several efficient algorithms for calculating robot dynamics were deduced. Results indicate that the proposed methods compare favourably with state-of-the-art methods.

Throughout this work MATLAB® was opted as the tool for implementing the algorithms, while the real-time virtual-reality simulations were carried out using the Virtual Experimentation Platform (V-REP). Using these tools several controllers, algorithms, techniques and simulations were applied and the results achieved were discussed. We conclude this study by identifying some of the issues associated with the artificial potential field method.

Keywords

Collision avoidance, collaborative robots, safety, Joint space inertia matrix, Coriolis matrix, Centrifugal matrix, time derivative of joint space inertia matrix.

Chapter 1

INTRODUCTION

Nowadays, robots are corner stones in modern factories. Their involvement in work space contributes to higher production capabilities, flexibility and accuracy. They are used extensively in manufacturing for pick and place operations, painting, welding, metal processing, and even in food industry. Owing to their accuracy, they are also becoming valuable tools for performing precise medical procedures. Also, with an ageing population, their existence in household environments is of an importance, while repetitive low level tasks can be delegated to them. This fact can be seen around us with robotic vacuum cleaners or lawn mowers. As such, robots are becoming more and more involved in our lives so that full human robot interaction and coexistence of robots in human-centered environments seems almost inevitable. This will require robots with better capabilities that are able to avert danger and ensure safety of the human coworker under all circumstances.

One of the first methodologies that can be utilized to achieve safe interaction with machines is through endowing them with collision avoidance capabilities. This thesis address this topic, and delves into the subject of collision avoidance for robotic manipulators in a human-robot shared workspace.

1.1 Collision avoidance

One of the first and the most important capabilities that robots need to be provided with are biological-like reflexes that allow them to circumvent obstacles and avoid collisions. This is extremely important in order to give robots more autonomy and minimum need for human intervention, especially when robots are operating in changing workspace and dynamic environment. Collision avoidance is also very important when humans and robots are collaborating with each other. Nowadays, it has been the norm to have boundaries and safety fences that stand between humans and industrial robots. The idea is to eliminate these boundaries and have humans and robots sharing the same workspace together, collaborating with each other, Figure 1.1. In such scenario robot's control system shall ensure the safety of the human coworker under all circumstances.

Several research efforts have focused on collision avoidance for industrial robots in gas and oil industry [3], their work is motivated by the fact that robots are vital if

Figure 1.1: Baxter robot with coworker in a factory, collision avoidance is of vital importance to guarantee the safety of the coworker.

Figure 1.2: A robot locate in potentially explosive environment, among pipes carrying gas [1].

Figure 1.3: Morphin map of Curiosity Mars rover, courtesy of Mars autonomy project.

humans want to tap into oil and gas reserves in remote regions where extremely harsh environments prevail, for example the reservoirs in the Barents Sea where temperatures can reach as low as $-50°C$. In such places working conditions for humans are unfeasible. Also, collision avoidance is important for robots operating in hazardous and harsh environments. Figure 1.2 demonstrates such concept with a robotic arm is operating a valve in explosive environment.

Apart from safety, autonomy is vital when robots are operating in remote places and unstructured environments. An example on this is the Curiosity Mars rover, which utilizes an obstacle avoidance algorithm called Morphin. This algorithm maintain a map of the environment, Figure 1.3, and based on this map Morphin recommends safe steering commands to the rover.

The problem of collision-avoidance is handled at one of two levels: global level addressed through planning and local level treated by low level control. The global solutions are high-level solutions that guarantee to find a collision-free path from the initial configuration to the final configuration, if such a path exists. These algorithms treat the problem in configuration space. In such case the manipulator and the environment need to be remapped to configuration space and a collision-free path is searched in the unoccupied portion of the configuration space. However, these algorithms are very costly in terms of computation, and because of that their use is not feasible for real-time application. On the other hand, the local reactive control is suitable for real-time implementation, since that its mathematical formulation is computationally efficient, so it can be embedded directly into the low level-control.

1.2 Tasks and objectives

The first objective is to implement light weight algorithms to calculate the dynamics of serially linked manipulators. When the modelling is complete, we implement two low-level robot controllers with collision avoidance capability for a collaborative robotic cell. The proposed robotic cell is commanded by a 6 DOF anthropomorphic manipulator and a human coworker operates in the same work area with the robot. The controller's function is to perform a predefined task while at the same time providing the robot with the capacity to perform real-time collision-avoidance in a dynamic and changing environment. For testing the proposed controllers and to assess its performance virtual-reality simulations were carried out using V-REP. To achieve these objectives the following topics were explored:

1. Comprehensive survey in the literature about robot's dynamics and collision avoidance for robotic manipulators.
2. Methodology to model the robot and the obstacles' geometry to calculate the distances between them.
3. Implementation of virtual reality robotics simulation program V-REP.
4. Modeling robot's dynamics and kinematics. Implementing the algorithms in MATLAB® and performing simulations.
5. Collision avoidance implementation based on potential field method.

1.3 Contribution

The thesis contribution is two-fold. The first pertains to the subject of dynamics of serially linked robots, while the other pertains to the subject of collision avoidance for robotic manipulators.

Regarding dynamics:

1. Implementation of several light weight algorithms to calculate the joint space, inertia matrix, Coriolis matrix, centrifugal matrix and the time derivative of the inertia matrix.

2. New algorithm for calculating the dynamics of serially linked robots. In which the algorithm proposed for calculating joint space inertia matrix (JSIM) for robots with high degrees of freedom achieves better efficiency over state-of-the-art.

3. Efficient algorithm for calculating rotation of inertia tensor of a rigid body, that achieves 14% better efficiency over the most efficient algorithm to our knowledge.

Regarding collision avoidance:

1. Developing fast algorithm based on optimization techniques for fast calculation of minimum distances between cylinders.

2. Proposing a novel collision avoidance controller based on linearising the inverse dynamics equation.

3. Novel solutions are proposed for solving some of the drawbacks associated with the artificial potential field method namely vibrations and joints limits avoidance.

1.4 Nomenclature

The notation used throughout this study is the same notation used in [4], and is described below:

1. Bold capital letters are used to denote matrices, $\mathbf{J}$ for the Jacobean matrix.

2. Bold and Italic small letters are used to denote vectors, $\boldsymbol{q}$ for the joints positions vector.

3. Italic small letters are used to denote scalars, q_j is the angular position of joint j.

4. Dot operators are used to denote time derivatives $\dot{q} = dq/dt$.

5. Column k of matrix $\mathbf{A}$ is denoted by $col_k(\mathbf{A})$.

6. Transpose of a matrix $\mathbf{A}$ is denoted by $\mathbf{A^T}$.

7. Row k of matrix A is denoted by $col_k(\mathbf{A^T})$.

8. Vector cross product is denoted by $\times$.

9. The skew symmetric operator of a vector $\boldsymbol{p}$ is notated by $\hat{\boldsymbol{p}}$. Where $\hat{\boldsymbol{p}}$ is used as the matrix representation of the cross product $\boldsymbol{p}\times$.

Chapter 2

DYNAMICS OF SERIALLY LINKED ROBOTS

2.1 Abstract

The evolution of advanced robots with higher degrees of freedom impose a need for calculating more complex dynamics. As a result, better efficiency in carrying out dynamics computations is becoming more important. In this study an efficient method for computing the dynamics for serially linked robots is addressed. We call this method the Geometric Dynamics Algorithm (GDA). GDA is non-symbolic, preserve simple formulation, and is convenient for numerical implementation. It allows the calculation of various quantities of robot dynamics while at the same time achieving computational efficiency. GDA-based algorithms are deduced to calculate (1) joint space inertia matrix (JSIM), (2) joint space Coriolis matrix, (3) joint space centrifugal matrix, and (4) the time derivative of joint space inertia matrix in $O(n^2)$. The proposed algorithms were implemented in MATLAB®. Results compare favorably with existing methods. One of the proposed algorithms, GDAHJ, achieves better performance over state-of-the-art when applied for high degree of freedom robots.

2.2 Introduction

Dynamics of robots is an important topic since that it is highly involved in their design, simulation and control. Owing to its importance this subject had been studied extensively in the past thirty years. Thus, several algorithms and methods had been developed to calculate robot dynamics. A comprehensive overview of the most important algorithms can be found in [5] and in [6]. Nevertheless, this subject remains till this day open for extensive research while every year there are new studies being published, methods and algorithms being proposed. In this section we give a brief introduction into the state-of-the-art regarding robot dynamics.

Robot dynamics can be described by one of two formulations:

1. Operational space formulation. In this formulation the dynamics equations are referenced to the manipulator end-effector. In a pioneering study this approach

was described and used to control PUMA600 robot [7]. It is also applied for the combined application of motion and force control [8]. Algorithms for efficient robot dynamics calculations based on operational space formulation are presented in [9] and [10].

2. Joint space formulation. This formulation describes the dynamics of robot in joint space. This formulation manifests the effect of the joints' positions, velocities and accelerations on the torques and vice-versa.

The mathematical formulation of the inverse dynamics in joint space [4, 11, 12, 13] is given by:

$$\boldsymbol{\tau} = \mathbf{A}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g} \tag{2.1}$$

Where $\boldsymbol{\tau}$ is the vector of robot's joints torques, $\boldsymbol{q}$ is the vector of joints positions, $\dot{\boldsymbol{q}}$ is the vector of joints' angular velocities, $\ddot{\boldsymbol{q}}$ is the vector of joints' angular accelerations, $\mathbf{A}(\boldsymbol{q})$ is joint space inertia matrix of the robot, $\mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the joint space Coriolis matrix of the robot, and $\boldsymbol{g}$ is the vector of joint's torques due to gravity. As described in [14] equation (2.1) can be extended to include contact forces, joints elasticity, friction, actuators inertias and dynamics. $\mathbf{A}(\boldsymbol{q})$ is an $n \times n$ matrix, in which n is the number of robot's joints considering that each joint has one degree of freedom (DOF), it is symmetric, positive definite and has the property of being a function of only joints' positions. $\mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is also an $n \times n$ matrix. It is a function of joints' positions and velocities, and describes the centrifugal and Coriolis effects on joints' torques.

One of the earliest methods used to deduce the equations of robot dynamics was the one based on Lagrangian formulation. This method is well described in the literature. A methodology for deducing the dynamics of gear-driven serially linked robot by using Lagrangian formulation is described in [15]. This study took into consideration the effects of the driving motors. The Lagrangian formulation is widely used as the bases for automatic generation of equations of robot dynamics in symbolic form. Most recent toolboxes for generating equations of robot dynamics using Lagrangian formulation are described in [16, 17].

The Lagrangian formulation is a straight forward approach that treats the robot as a whole and utilizes its Lagrangian, a function that describes the energy of the mechanical system:

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \tag{2.2}$$

Where $\mathcal{L}$ is the Lagrangian function, $\mathcal{T}$ is the kinetic energy and $\mathcal{U}$ is the potential energy. The function described previously is formulated in terms of the generalized coordinates $\boldsymbol{q}$. By differentiating that function we can derive an expression of the associated generalized forces $\boldsymbol{v}$, as in:

$$\boldsymbol{v} = \frac{d}{dt}\left(\frac{\delta \mathcal{L}}{\delta \dot{\boldsymbol{q}}}\right)^{\mathrm{T}} - \left(\frac{\delta \mathcal{L}}{\delta \boldsymbol{q}}\right)^{\mathrm{T}} \tag{2.3}$$

Even though the Lagrangian formulation can be considered as a straight forward approach, the method requires partial differentiation, and despite the fact that symbolic manipulation methods have been utilized to perform the differentiation [18], the

method still lacks the efficiency in terms of execution-time. This can be clearly noticed when the robot presents a high number of DOF as noted in [12] and most remarkably in [19], where the author performed comparison of execution-times required to run simulations based on dynamical models derived by Newton-Euler recursive technique and Euler-Lagrange technique. It was reported execution-times difference of order of magnitude which clearly put the case in favor of the Newton-Euler recursion method.

The formulation of robot-specific dynamics using Kane's dynamical equations is in [20]. In this study the authors argue that using Lagrange method to compute dynamics produces huge equations resulting in slow execution and costly computations, while the Recursive Newton-Euler is a generalized method that might perform unnecessary calculations on specific robot. Thus, a faster execution algorithm with less computational-cost could be achieved if robot-specific equations are carefully deduced. The study elaborates in step by step manner the methodology for deriving dynamics equations of Stanford manipulator starting from Kane's dynamical equations. Nevertheless, the method requires a knowledgeable analyst to take on a pencil and paper in hand and work out the equations of a specific robot. A comprehensive review of Kane's equations and Gibbs-Appell equations is in [21].

In [22] the authors presented an algorithm for calculating JSIM, Coriolis matrix and the vector of gravity torques. In the algorithm proposed, several parameters are precalculated off-line, and others are calculated on-line, afterwords the inertia and Coriolis matrices are deduced, the computational complexity of the presented algorithm was of $O(n^3)$.

A computationally efficient Newton-Euler recursive method is described in [23]. This method is performed in two phases: the first phase (forward propagation) during which the accelerations and velocities of robot links are calculated, and the second phase (backward propagation) where torques and forces are calculated. The method proved to be very efficient for calculating the inverse dynamics. However, the calculations are carried out implicitly such that the inertia matrix cannot be retrieved directly. It is shown in [24] that the inertia matrix $\mathbf{A}(\boldsymbol{q})$ can be calculated from the model of the inverse dynamics by assigning a unit value to one element of the joints' accelerations vector and assigning a zero value to the remaining elements, including the joints' velocities and the gravity term. In such scenario the associated column of the inertia matrix can be calculated, and by iterating the procedure through all of the elements of the joints' acceleration vector the inertia matrix is achieved. This method was later re-named composite-rigid-body algorithm (CRBA), by Featherstone [25]. Using CRBA to calculate the inertia matrix proved to be computationally efficient, especially if the calculations are performed in links-attached local frames. Computer code of the algorithm based on 6D or spatial vectors algebra is available in [25]. A comprehensive review of spatial vectors and Plücker basis is in [26] and in chapter 2 of [27].

In [28] the author reformulated robot dynamics using Riemannian geometry. Several algorithms were presented. Based on Newton-Euler formulation a recursive algorithm of $O(n)$ for calculating the inverse dynamics was introduced. Based on Lagrangian formulation, algorithms for deducing closed form equations of dynamics, JSIM and Coriolis matrix were presented. In-addition the paper laid down a frame-work for calculating the derivative of dynamics, which is of importance when performing dynamic optimization. Building on [28], in [29] the author introduced coordinate in-

variant formulation of the dynamics. In those two studies algorithms were proposed, but no quantitative measure about their efficiency, apart from the case of the recursive algorithm, was given.

In [30] the dynamics was formulated using 4×4 matrices. This method can be considered as an extension to homogeneous transformation matrices proposed by Denavit and Hartenberg. Thus, five new 4×4 matrices were introduced, to quantify velocity, acceleration, momentum, actions, and the inertia. In [31] two software packages in addition to applications of the proposed approach were presented, though no quantitative measure on the computational efficiency of the proposed method was given.

Calculating Coriolis matrix is important for some control applications, for example in passivity-based control as noted in [32]. In [33], the Coriolis matrix, $\mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, according to the notation of that paper, was used for calculating collision detection signal. In that study the Coriolis matrix transpose appears in the term $\mathbf{C}^{\mathrm{T}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ of equation (22), as such Coriolis matrix needed to be calculated explicitly in real time for performing collision identification and real time reaction control.

Coriolis matrix can be deduced in a symbolic form by utilizing Euler-Lagrange formulation through partial differentiation and by utilizing Christoffel symbol of the first kind. However, as discussed in the case of inertia matrix, the attained formulas are slow in execution. Other approaches have been proposed, most recently in [34], where Coriolis matrix was factorized in a closed form expression of kinematic matrices and their derivatives. Later, the method was extended in [35] in order to include the coupling effect for a geared serial robot. In [32] Coriolis and centrifugal terms were factorized using a modified recursive Newton-Euler method. The paper focused on computational efficiency.

In this study we propose efficient algorithms for calculating: joint space inertia matrix (JSIM), Coriolis matrix, centrifugal matrix, and the time derivative of JSIM, (TD-JSIM), for serially linked robots. In the proposed method dynamics quantities are calculated as contributions of frames' effects in what we call the frame injection effect. The principal of frame injection effect is described in this study. The novelty of the proposed method is in its structure which describes and preserves the effect of each joint's velocity and acceleration on links' dynamics. As such algorithms can be deduced for representing dynamical quantities in a mathematical form that resembles the equation of inverse dynamics, using this representation computational efficiency is achieved while maintaining simple algorithms. To support our claim, we give the following example: it has been noted in [32] that in [33] the authors had to reformulate the equation of the rate of the generalized momentum in order to avoid numerical differentiation of JSIM. This study shows that by using the proposed method deducing an efficient algorithm $O(n^2)$ for calculating the time derivative of JSIM is viable and easy to implement. The proposed method is generic, simple and relies completely on vector operations, as such no symbolic operations, and the inconvenience they convey, are required. When executed on single processor this method is of an $O(n^2)$, but the algorithm can be executed in parallel, as such we can reap the power of multiprocessor machines by using threading. In this scenario the main program will spawn n threads, equal to the number of links of the robot, each thread will be used to calculate the dynamical model of one of the links.

In addition to GDA, we present a novel method (GDAHJ) for calculating joint

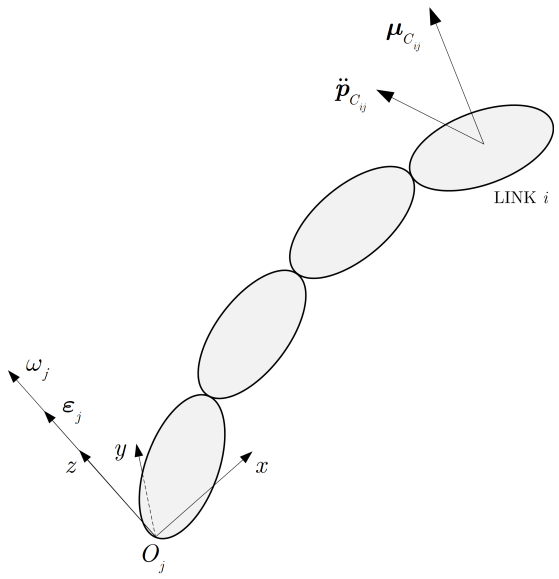

Figure 2.1: Inertial moment $\boldsymbol{\mu}_{\mathrm{C}ij}$ and linear acceleration $\ddot{\boldsymbol{p}}_{\mathrm{C}ij}$ of centre of mass of link i transferred by frame j

space inertia matrix for robots with high degrees of freedom. Using this method for computing JSIM for articulated bodies with high degrees of freedom achieves better efficiency over state-of-the-art method, the famous CRBA. This increase in efficiency is achieved through minimizing the number of operations that has $O(n^2)$ computational complexity. While in the proposed algorithm, the number of computations associated with the quadratic terms are reduced to the minimum value possible, from $16n^2$ in the case of CRBA to $5.5n^2$ for the proposed algorithm.

For evaluating the performance of the proposed algorithms, a comparison with existing state-of-the-art algorithms was performed and the acquired results are discussed in section 2.6.

2.3 Theory and principals

The proposed algorithm depends on what we call the frame injection effect, Figure 2.1, in which each frame j attached to joint j will transfer to link i a linear acceleration into its centre of mass and an inertial moment around its centre of mass. In this study we notate them by $\ddot{\boldsymbol{p}}_{\mathrm{C}ij}$ and $\boldsymbol{\mu}_{\mathrm{C}ij}$, respectively. This transfer is due to the rotational effect of joint j around its axes of rotation, or the z axis of frame j according to modified Denavit Hartenberg (MDH) designation. This cause and effect relationship between frame j and link i is referred to by the subscript ij in $\ddot{\boldsymbol{p}}_{\mathrm{C}ij}$ and $\boldsymbol{\mu}_{\mathrm{C}ij}$, while the C in the subscript is used to refer to the mass centre of link i. The same subscript notation will hold throughout this study for denoting frame-link interaction of cause-and-effect unless stated otherwise.

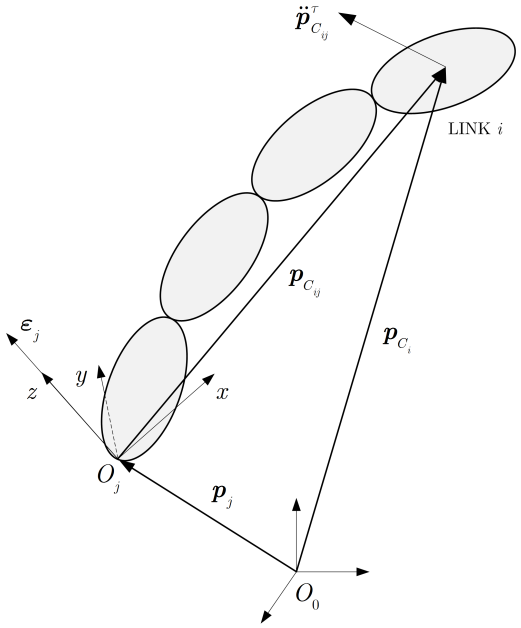

Figure 2.2: Tangential acceleration of centre of mass of link i transferred by frame j

2.3.1 Link's acceleration due to the single-frame rotation

As we described previously the algorithm proposed relies on what we call the frame injection effect, while it can be proved that each frame j transfers to link i three acceleration vectors tangential acceleration, normal acceleration and Coriolis acceleration. The first of which is shown in Figure 2.2, it is due to the angular acceleration of frame j, and it can be calculated from:

$$\ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij} = \boldsymbol{\varepsilon}_j \times \boldsymbol{p}_{\mathrm{C}ij} \tag{2.4}$$

Where $\ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij}$ is the tangential acceleration of the centre of mass of link i due to the rotation of frame j, the symbol $\times$ is used to denote the cross product (the same notation of the cross product will hold throughout this study) and $\boldsymbol{p}_{\mathrm{C}ij}$ is the vector connecting the origin of frame j and the centre of mass of link i. $\boldsymbol{\varepsilon}_j$ is the angular acceleration of link j, and it is given by:

$$\boldsymbol{\varepsilon}_j = \ddot{q}_j \boldsymbol{k}_j \tag{2.5}$$

Where $\boldsymbol{k}_j$ is the unit vector associated with the z axis of joint j, and $\ddot{q}_j$ is the angular acceleration of that joint.

Then there is the normal acceleration: while each frame j transfers to link i a normal acceleration due to its rotation as shown in Figure 2.3, and it is given by:

$$\ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} = \boldsymbol{\omega}_j \times (\boldsymbol{\omega}_j \times \boldsymbol{p}_{\mathrm{C}ij}) \tag{2.6}$$

Where $\boldsymbol{\omega}_j$ is the angular velocity of link j due to the rotational effect of joint j, and it is given by:

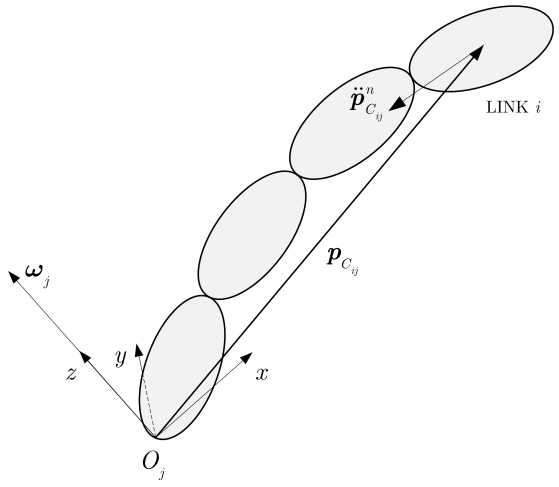

Figure 2.3: Normal acceleration of centre of mass of link i transferred by frame j

$$\boldsymbol{\omega}_j = \boldsymbol{k}_j \dot{q}_j \tag{2.7}$$

We can rewrite the equation of the normal acceleration transferred to link i due to frame j by the following:

$$\ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} = \boldsymbol{k}_j \times (\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij})\dot{q}_j^2 \tag{2.8}$$

The third acceleration transferred is Coriolis acceleration, as shown in Figure 2.4, while each frame j transfers to link i Coriolis acceleration $\ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij}$:

$$\ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij} = 2\boldsymbol{\omega}_j \times \boldsymbol{v}^{r}_{\mathrm{C}ij} \tag{2.9}$$

Where $\boldsymbol{\omega}_j$ is as described previously in equation (2.7), and $\boldsymbol{v}^{r}_{\mathrm{C}ij}$ is the velocity transferred to the centre of mass of link i from frames $j+1$ up to frame i, while the r in the superscript is to denote that this is a relative velocity, and C in the subscript is used to refer to the mass centre of link i, $\boldsymbol{v}^{r}_{\mathrm{C}ij}$ can be calculated from:

$$\boldsymbol{v}^{r}_{\mathrm{C}ij} = \sum_{k=j+1}^{i} \boldsymbol{\omega}_k \times \boldsymbol{p}_{\mathrm{C}ik} \tag{2.10}$$

The total linear acceleration transferred by frame j to the centre of mass of link i is given by:

$$\ddot{\boldsymbol{p}}_{\mathrm{C}ij} = \ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij} \tag{2.11}$$

2.3.2 Link's inertial moment due to single-frame effect

It can be proved that each frame j will transfer to link i three inertial moments, the first of the inertial moments transferred is due to angular acceleration of frame j and it is given by:

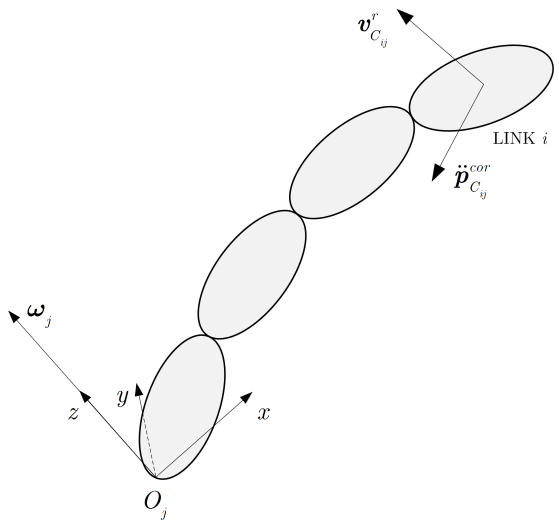

Figure 2.4: Coriolis acceleration of centre of mass of link i transferred from frame j

$$\boldsymbol{\mu}^{\tau}_{\mathrm{C}ij} = (\mathbf{R}_i \mathbf{I}^i_i \mathbf{R}^{\mathrm{T}}_i)\boldsymbol{\varepsilon}_j \tag{2.12}$$

While $\boldsymbol{\mu}^{\tau}_{\mathrm{C}ij}$ is the moment transferred by frame j into link i due to frame's j angular acceleration, $\mathbf{R}_i$ is the rotation matrix of frame i in relation to base frame, and $\mathbf{I}^i_i$ is 3×3 inertial tensor of link i around its centre of mass represented in frame i. Calculating the similarity transform $\mathbf{R}_i \mathbf{I}^i_i \mathbf{R}^{\mathrm{T}}_i$ in the conventional way is expensive, to achieve the best possible performance we propose a novel way for performing this type of computation, in Appendix I. The proposed algorithm compares favourably with state-of-the-art method, and delivers 14% better performance.

The second inertial moment transferred from frame j to link i is due to centrifugal effect:

$$\boldsymbol{\mu}^{n}_{\mathrm{C}ij} = \frac{1}{2}(\mathbf{L}_i \boldsymbol{\omega}_j) \times \boldsymbol{\omega}_j \tag{2.13}$$

Where $\mathbf{L}_i$ is 3×3 matrix that is calculated from:

$$\mathbf{L}_i = \mathbf{R}_i(\mathrm{tr}(\mathbf{I}^i_i)\mathbf{1}_3 - 2\mathbf{I}^i_i)\mathbf{R}^{\mathrm{T}}_i \tag{2.14}$$

The subscript in $\mathbf{L}_i$ is to notate that the matrix calculated pertains to link i. While $\mathrm{tr}(\mathbf{I}^i_i)$ is the trace of the inertial tensor, and $\mathbf{1}_3$ is the identity matrix.

The third inertial moment transferred from frame j to link i is due to Coriolis effect:

$$\boldsymbol{\mu}^{cor}_{\mathrm{C}ij} = (\mathbf{L}_i \boldsymbol{\omega}_j) \times \boldsymbol{\omega}^r_{ij} \tag{2.15}$$

Where $\boldsymbol{\omega}^r_{ij}$ can be calculated from:

$$\boldsymbol{\omega}^r_{ij} = \sum_{k=j+1}^{i} \boldsymbol{\omega}_k \tag{2.16}$$

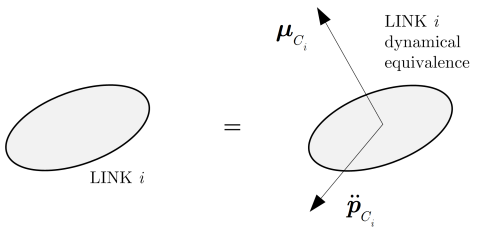

Figure 2.5: Dynamical representation of each link.

Thus, the total inertial moment transferred to link i around its centre of mass due to the rotational effect of frames j is given by:

$$\boldsymbol{\mu}_{\mathrm{C}ij} = \boldsymbol{\mu}^{\tau}_{\mathrm{C}ij} + \boldsymbol{\mu}^{n}_{\mathrm{C}ij} + \boldsymbol{\mu}^{cor}_{\mathrm{C}ij} \tag{2.17}$$

2.3.3 Dynamical representation of a single-link

Each link i can be represented dynamically by an equivalent acceleration of its centre of mass $\ddot{\boldsymbol{p}}_{\mathrm{C}i}$ and an inertial moment around its centre of mass $\boldsymbol{\mu}_{\mathrm{C}i}$ as shown in Figure 2.5.
The total linear acceleration of the centre of mass of link i is given by the summation of all of the linear accelerations transferred by all of the serially connected frames indexed j, starting from frame 1 up to frame i:

$$\ddot{\boldsymbol{p}}_{\mathrm{C}i} = \sum_{j=1}^{i} \ddot{\boldsymbol{p}}_{\mathrm{C}ij} \tag{2.18}$$

Substituting $\ddot{\boldsymbol{p}}_{\mathrm{C}ij}$ with its value from (2.11) we get:

$$\ddot{\boldsymbol{p}}_{\mathrm{C}i} = \sum_{j=1}^{i} \ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij} \tag{2.19}$$

From the previous equation we notice that the total acceleration of the centre of mass of link i can be rewritten in a form similar to the equation of inverse dynamics by using a matrix vector notation as in the following:

$$\ddot{\boldsymbol{p}}_{\mathrm{C}i} = \mathbf{C}_i \ddot{\boldsymbol{q}} + \mathbf{D}_i \dot{\boldsymbol{q}} \tag{2.20}$$

Where $\mathbf{C}_i$ is $3 \times n$ matrix, and the j^{th} column vector of this matrix is given by:

$$\mathrm{col}_j(\mathbf{C}_i) = \frac{\ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij}}{\ddot{q}_j} = \boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij} \tag{2.21}$$

The subscript i attached to the matrix $\mathbf{C}_i$ is used to notate that the matrix pertains to link i, while the robot's links model has n of $\mathbf{C}$ matrices each pertains to one of robot's links. $\mathbf{D}_i$ in equation (2.20) is also $3 \times n$ matrix, while the subscript i in $\mathbf{D}_i$ is used as before. $\mathbf{D}_i$ columns can be calculated from:

$$\mathrm{col}_j(\mathbf{D}_i) = \boldsymbol{k}_j \times (\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij})\dot{q}_j + 2\boldsymbol{k}_j \times \boldsymbol{v}^r_{\mathrm{C}ij} \tag{2.22}$$

Using the same reasoning applied for calculating $\ddot{\boldsymbol{p}}_{\mathrm{C}i}$, we can calculate the total inertial moment transferred to link i by:

$$\boldsymbol{\mu}_{\mathrm{C}i} = \sum_{j=1}^{i} \boldsymbol{\mu}^{\tau}_{\mathrm{C}ij} + \boldsymbol{\mu}^{n}_{\mathrm{C}ij} + \boldsymbol{\mu}^{cor}_{\mathrm{C}ij} \tag{2.23}$$

Again we notice that the joints' accelerations vector $\ddot{\boldsymbol{q}}$ contribution to the inertial moment is associated only with $\boldsymbol{\mu}^{\tau}_{\mathrm{C}ij}$, while the joints velocities contributions are associated with the other two terms.

As such we can distinguish and show the effects of joints' acceleration vector $\ddot{\boldsymbol{q}}$, and joints' velocities vector $\dot{\boldsymbol{q}}$ on $\boldsymbol{\mu}_{\mathrm{C}i}$ by rearranging equation (2.23) using a matrix vector notation as in the following:

$$\boldsymbol{\mu}_{\mathrm{C}i} = \mathbf{U}_i\ddot{\boldsymbol{q}} + \mathbf{V}_i\dot{\boldsymbol{q}} \tag{2.24}$$

Where $\mathbf{U}_i$ is $3 \times n$ matrix, each column vector of this matrix is given by:

$$\mathrm{col}_j(\mathbf{U}_i) = (\mathbf{R}_i\mathbf{I}^i_i\mathbf{R}^{\mathrm{T}}_i)\boldsymbol{k}_j \tag{2.25}$$

$\mathbf{R}_i$ is the rotation matrix defining frame i in relation to the base frame, and $\boldsymbol{k}_j$ is the unit vector of the z axis attached to joint j represented in base frame. $\mathbf{V}_i$ is $3 \times n$ matrix, each column vector of this matrix is given by:

$$\mathrm{col}_j(\mathbf{V}_i) = \frac{1}{2}(\mathbf{L}_i\boldsymbol{\omega}_j) \times \boldsymbol{k}_j + (\mathbf{L}_i\boldsymbol{k}_j) \times \boldsymbol{\omega}^r_{ij} \tag{2.26}$$

As a result each link is represented by a linear acceleration of its centre of mass, and an inertial moment around its centre of mass, the mathematical equation is formulated in a way that explicitly express the effects of $\ddot{\boldsymbol{q}}$ and $\dot{\boldsymbol{q}}$. Thus, the dynamics of each link is defined completely by four $3 \times n$ matrices $\mathbf{U}_i$,$\mathbf{C}_i$, $\mathbf{D}_i$ and $\mathbf{V}_i$ as such we have clear representation of links' dynamics as linearised function of joints accelerations and velocities.

2.3.4 Moment acting on a joint due to robot dynamics

The total moment acting on any joint j due to robot dynamics, $\boldsymbol{\mu}_j$ in Figure 2.6, can be calculated from:

$$\boldsymbol{\mu}_j = \sum_{i=j}^{n} \boldsymbol{\mu}_{\mathrm{C}i} + m_i\boldsymbol{p}_{\mathrm{C}ij} \times \ddot{\boldsymbol{p}}_{\mathrm{C}i} \tag{2.27}$$

Which can be rearranged in a form resembling the inverse dynamics equation as in the following:

$$\boldsymbol{\mu}_j = \mathbf{G}_j\ddot{\boldsymbol{q}} + \mathbf{H}_j\dot{\boldsymbol{q}} \tag{2.28}$$

While $\mathbf{G}_j$ is $3 \times n$ matrix, each column k of this matrix is given by:

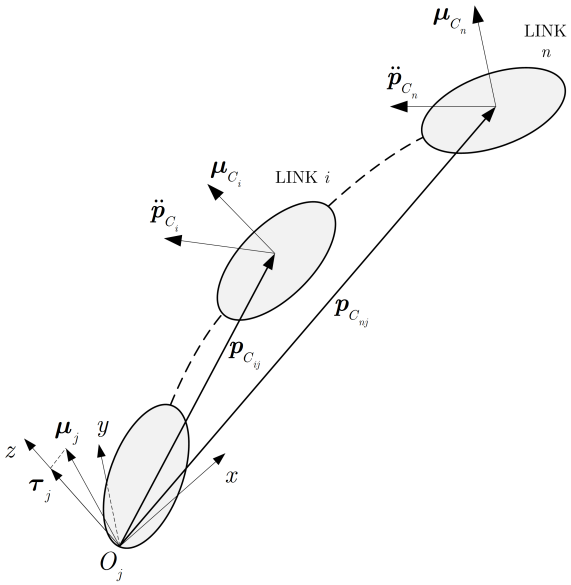

Figure 2.6: Moment acting on joint j due to robot's motion

$$\mathrm{col}_k(\mathbf{G}_j) = \sum_{i=j}^{n} \mathrm{col}_k(\mathbf{U}_i) + m_i \boldsymbol{p}_{Cij} \times \mathrm{col}_k(\mathbf{C}_i) \tag{2.29}$$

In a similar way, each column k of matrix $\mathbf{H}_j$ can be calculated from the following:

$$\mathrm{col}_k(\mathbf{H}_j) = \sum_{i=j}^{n} \mathrm{col}_k(\mathbf{V}_i) + m_i \boldsymbol{p}_{Cij} \times \mathrm{col}_k(\mathbf{D}_i) \tag{2.30}$$

Subscript j in $\mathbf{H}_j$ and $\mathbf{G}_j$ is to denote that these matrices are associated with joint j.

2.3.5 Joint space inertia, Coriolis & Centrifugal matrices

The torque acting on joint k due to robot dynamics is calculated from projecting $\boldsymbol{\mu}_k$, derived in the previous section, onto the z axis of joint k as in:

$$\boldsymbol{\tau}_k = \boldsymbol{k}_k^{\mathrm{T}} \boldsymbol{\mu}_k \tag{2.31}$$

Each row k of JSIM, or $\mathrm{col}_k(\mathbf{A}^{\mathrm{T}})$, is calculated from:

$$\mathrm{col}_k(\mathbf{A}^{\mathrm{T}}) = \boldsymbol{k}_k^{\mathrm{T}} \mathbf{G}_k \tag{2.32}$$

Using the same notion, each row k of Coriolis matrix $\mathbf{B}$, or $\mathrm{col}_k(\mathbf{B}^{\mathrm{T}})$ can be calculated from:

$$\mathrm{col}_k(\mathbf{B}^{\mathrm{T}}) = \boldsymbol{k}_k^{\mathrm{T}} \mathbf{H}_k \tag{2.33}$$

Thus, the inverse dynamics equation of the robot defined by the inertial matrix $\mathbf{A}(\boldsymbol{q})$ and Coriolis matrix $\mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, has been derived.

Using the same principals developed in this section the joint space centrifugal matrix can be calculated. This is done by considering only the terms resulting from Centrifugal accelerations during the calculation of matrices $\mathbf{D}_i$, $\mathbf{V}_i$ and $\mathbf{H}_j$. Based on the equations presented in this section three computationally efficient MATLAB® functions were implemented for calculating JSIM, in addition to joint space Centrifugal and Coriolis matrices.

2.3.6 Calculating the time derivative of inertia matrix

The time derivative of JSIM shows up as a by-product when calculating the time derivative of the generalized momentum in [33], which has applications in collision detection, while as noted in [32] the author had to change the formulation of the equation describing the rate of change of the generalized momentum in order to avoid the numerical differentiation of JSIM since calculating it numerically is expensive $O(n^3)$. Thus, the use of TD-JSIM is being avoided, a disadvantage since that it has a favourable property of being symmetric. In this section we describe the methodology for deducing an efficient and fast algorithm of $O(n^2)$ for calculating TD-JSIM on the bases of the frame injection principal. The proposed algorithm was implemented in MATLAB®. The computational cost of this algorithm and a comparison with numerically differentiating JSIM is presented in operation count section of this study.

It has been shown previously in equation (2.32) that each row of JSIM is given by:

$$\mathrm{col}_k(\mathbf{A}^{\mathrm{T}}) = \boldsymbol{k}_k^{\mathrm{T}}\mathbf{G}_k$$

Thus the time derivative of matrix $\mathbf{A}$ is defined by the time derivative of its rows as in:

$$\mathrm{col}_k(\dot{\mathbf{A}}^{\mathrm{T}}) = \dot{\boldsymbol{k}}_k^{\mathrm{T}}\mathbf{G}_k + \boldsymbol{k}_k^{\mathrm{T}}\dot{\mathbf{G}}_k \tag{2.34}$$

Since $\boldsymbol{k}_k^{\mathrm{T}}$ is of a constant magnitude then its time derivative is given by:

$$\dot{\boldsymbol{k}}_k^{\mathrm{T}} = (\boldsymbol{\omega}_k^0 \times \boldsymbol{k}_k)^{\mathrm{T}} \tag{2.35}$$

Where $\boldsymbol{\omega}_k^0$ is the angular velocity of frame k relative to frame k. Since that the derivative of matrix $\mathbf{G}_k$ can be defined by the derivative of its columns, then by considering equation (2.29), each column of $\dot{\mathbf{G}}_k$ is calculated from:

$$\begin{aligned}\mathrm{col}_j(\dot{\mathbf{G}}_k) = \sum_{i=k}^{n} \mathrm{col}_j(\dot{\mathbf{U}}_i) + m_i\dot{\boldsymbol{p}}_{Cik} \times \mathrm{col}_j(\mathbf{C}_i) \\ + m_i\boldsymbol{p}_{Cik} \times \mathrm{col}_j(\dot{\mathbf{C}}_i)\end{aligned} \tag{2.36}$$

While it can be shown that the derivative terms inside the summation of the previous equation are equal to:

$$\mathrm{col}_j(\dot{\mathbf{U}}_i) = \boldsymbol{\omega}_i^0 \times (\mathbf{R}_i\mathbf{I}_i^i\mathbf{R}_i^{\mathrm{T}}\boldsymbol{k}_j) - (\mathbf{R}_i\mathbf{I}_i^i\mathbf{R}_i^{\mathrm{T}})((\boldsymbol{\omega}_i^0 - \boldsymbol{\omega}_j^0) \times \boldsymbol{k}_j) \tag{2.37}$$

It is worth noticing that the term $(\boldsymbol{\omega}_i^0 - \boldsymbol{\omega}_j^0)$ of the previous equation is the same physical quantity $\boldsymbol{\omega}_{ij}^0$ in (2.16).

$$\dot{\boldsymbol{p}}_{Cik} = \boldsymbol{v}_{Ci} - \boldsymbol{v}_k \tag{2.38}$$

Where $\boldsymbol{v}_{Ci}$ is the linear velocity of the centre of mass of link i and $\boldsymbol{v}_k$ is the linear velocity of origin of frame k.

$$\mathrm{col}_j(\dot{\mathbf{C}}_i) = (\boldsymbol{\omega}_j^0 \times \boldsymbol{k}_j) \times \boldsymbol{p}_{\mathrm{C}ij} + \boldsymbol{k}_j \times (\boldsymbol{v}_{\mathrm{C}i} - \boldsymbol{v}_j) \tag{2.39}$$

Where $\boldsymbol{v}_j$ is the linear velocity of origin of frame j. Thus, the TD-JSIM can be calculated. The equations of this section were used to write an efficient $O(n^2)$ algorithm for calculating JSIM. We also want to note here that with slight modification of the algorithm proposed the term $\mathbf{C}^{\mathrm{T}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ in equation (22) of [33] can be calculated numerically and efficiently from the relation

$$\mathbf{C}^{\mathrm{T}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \dot{\mathbf{A}}\dot{\boldsymbol{q}} - \boldsymbol{b}$$

Where $\boldsymbol{b}$ is Coriolis vector. This calculation can be done efficiently, by taking an advantage of the fact that several parameters of robot dynamics are calculated at the same time when $\dot{\mathbf{A}}$ is being calculated, thus $\boldsymbol{b}$ can be calculated as a by-product while calculating $\dot{\mathbf{A}}$, the computational cost of the proposed algorithms is described in operation count section.

2.4 JSIM for hyper-joint manipulators

In this section we present a novel method, GDAHJ, for calculating joint space inertia matrix for robots with high DOF. Using this method for computing JSIM for articulated bodies with high degrees of freedom leads better efficiency over state-of-the-art method, CRBA. This increase in efficiency is achieved through minimizing the number of operations that has $O(n^2)$ computational complexity.

As described in section 3.2 of [5], column j of the joint space inertia matrix can be interpreted as: the torques acting on the various joints of the robot, due to the unit acceleration of joint j, giving that the angular velocities of all of the joints are equal to zero. In Figure 2.7 we show the free body diagram of one link of the robot, with the inertial moments and inertial forces acting on it.

Following the previous definition of column j of JSIM, we can calculate that column as the following: (1) choose a joint j, (2) write the balance equation of a link i from the robot. By referring to Figure 2.7, the balance equation of link i:

$$\boldsymbol{\mu}_{i,j} = \boldsymbol{\mu}_{i+1,j} + (\mathbf{R}_i \mathbf{I}_i^i \mathbf{R}_i^{\mathrm{T}})\boldsymbol{k}_j \ddot{q}_j + m_i \hat{\boldsymbol{p}}_{\mathrm{C}ii}(\ddot{q}_j \boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij}) +$$

$$\hat{\boldsymbol{l}}_i \sum_{k=i+1}^{n} m_k (\ddot{q}_j \boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}kj}) \tag{2.40}$$

Where $\boldsymbol{\mu}_{i,j}$ is the total moment acting on joint i due to the acceleration of joint j only. $\boldsymbol{l}_i$ is the vector connecting the origin of frame i to the proceeding frame's origin, the little hat notation above the vector is used to denote the skew symmetric operator

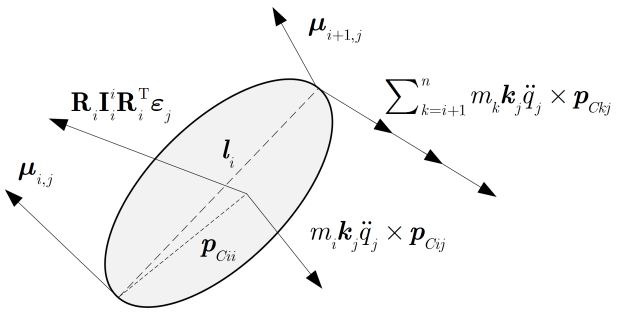

Figure 2.7: Inertial forces and moments acting on link i due to angular acceleration of joint j.

associated with that vector. From the definition given in the previous section of column i, we substitute $\ddot{q}_j$ by its value $\ddot{q}_j = 1$. Then the modified balance equation is:

$$\boldsymbol{\mu}_{i,j} = \boldsymbol{\mu}_{i+1,j} + (\mathbf{R}_i\mathbf{I}_i^i\mathbf{R}_i^{\mathrm{T}})\boldsymbol{k}_j + m_i\hat{\boldsymbol{p}}_{\mathrm{C}ii}(\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij})$$

$$+ \hat{\boldsymbol{l}}_i \sum_{k=i+1}^{n} m_k(\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}kj}) \tag{2.41}$$

While:

$$\boldsymbol{p}_{\mathrm{C}ij} = \boldsymbol{p}_{\mathrm{C}i} - \boldsymbol{p}_j \tag{2.42}$$

And:

$$\boldsymbol{p}_{\mathrm{C}kj} = \boldsymbol{p}_{\mathrm{C}k} - \boldsymbol{p}_j \tag{2.43}$$

We substitute the values of $\boldsymbol{p}_{\mathrm{C}ij}$ and $\boldsymbol{p}_{\mathrm{C}kj}$ into (2.41), and we fix:

$$\boldsymbol{\mu}_{i,j} = \boldsymbol{\mu}_{i+1,j} + \left(\mathbf{R}_i\mathbf{I}_i^i\mathbf{R}_i^{\mathrm{T}} - m_i\hat{\boldsymbol{p}}_{\mathrm{C}ii}\hat{\boldsymbol{p}}_{\mathrm{C}i} - \hat{\boldsymbol{l}}_i\left(\sum_{k=i+1}^{n} m_k\hat{\boldsymbol{p}}_{\mathrm{C}k}\right)\right)\boldsymbol{k}_j$$

$$- \left(m_i\boldsymbol{p}_{\mathrm{C}ii} + (\sum_{k=i+1}^{n} m_k)\boldsymbol{l}_i\right) \times (\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{2.44}$$

We define the vector $\boldsymbol{\eta}_i$ by:

$$\boldsymbol{\eta}_i = m_i\boldsymbol{p}_{\mathrm{C}ii} + (\sum_{k=i+1}^{n} m_k)\boldsymbol{l}_i \tag{2.45}$$

And we define the matrix operator $\boldsymbol{\kappa}_i$ by:

$$\boldsymbol{\kappa}_i = -m_i\hat{\boldsymbol{p}}_{\mathrm{C}ii}\hat{\boldsymbol{p}}_{\mathrm{C}i} - \hat{\boldsymbol{l}}_i\left(\sum_{k=i+1}^{n} m_k\hat{\boldsymbol{p}}_{\mathrm{C}k}\right) \tag{2.46}$$

Algorithm 2.1 Calculating joint space inertia matrix entries.

For $i = 1 : n$
  For $j = 1 : i$
% calculating $\mathbf{A}_{i,j}$ will require two vector inner products and one
% scalar addition resulting in total cost of $(3n^2 + 3n)m + (2.5n^2 + 2.5n)a$
    $\mathbf{A}_{i,j} = \boldsymbol{k}_j^{\mathrm{T}} \boldsymbol{d}_i + \boldsymbol{t}_j^{\mathrm{T}} \boldsymbol{y}_i$
    $\mathbf{A}_{j,i} = \mathbf{A}_{i,j}$
  End
End

Then we write:

$$\boldsymbol{\mu}_{i,j} = \boldsymbol{\mu}_{i+1,j} + \left(\mathbf{R}_i \mathbf{I}_i^i \mathbf{R}_i^{\mathrm{T}} + \boldsymbol{\kappa}_i\right) \boldsymbol{k}_j - (\boldsymbol{\eta}_i) \times (\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{2.47}$$

By performing a recursion on previous equation from link n to link i, and noticing that $\boldsymbol{\mu}_{n+1,j} = \mathbf{0}$ we get:

$$\boldsymbol{\mu}_{i,j} = \left(\sum_{k=i}^{n} \left(\mathbf{R}_k \mathbf{I}_k^k \mathbf{R}_k^{\mathrm{T}} + \boldsymbol{\kappa}_k\right)\right) \boldsymbol{k}_j - \left(\sum_{k=i}^{n} \boldsymbol{\eta}_k\right) \times (\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{2.48}$$

To hide the complexity in the previous equation, we denote the terms between parenthesis by:

$$\boldsymbol{b}_i = \left(\sum_{k=i}^{n} \boldsymbol{\eta}_k\right) \tag{2.49}$$

And

$$\mathbf{D}_i = \sum_{k=i}^{n} \left(\mathbf{R}_k \mathbf{I}_k^k \mathbf{R}_k^{\mathrm{T}} + \boldsymbol{\kappa}_k\right) \tag{2.50}$$

Substituting (2.49) and (2.50) in (2.48) yields:

$$\boldsymbol{\mu}_{i,j} = \mathbf{D}_i \boldsymbol{k}_j - \hat{\boldsymbol{b}}_i (\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{2.51}$$

For calculating the (i, j) entry of JSIM, $\mathbf{A}_{i,j}$, we project $\boldsymbol{\mu}_{i,j}$ on the z axes of joint i, or in other words we multiply (2.51) by the unit vector $\boldsymbol{k}_i^{\mathrm{T}}$:

$$\mathbf{A}_{i,j} = \boldsymbol{k}_i^{\mathrm{T}} \boldsymbol{\mu}_{i,j} = \boldsymbol{k}_i^{\mathrm{T}} \mathbf{D}_i \boldsymbol{k}_j - \boldsymbol{k}_i^{\mathrm{T}} \hat{\boldsymbol{b}}_i (\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{2.52}$$

By noticing that each entry i, j of the JSIM, or $\boldsymbol{k}_i^{\mathrm{T}} \boldsymbol{\mu}_{i,j}$ is a scalar, then we can transpose the previous equation without loss of generality:

$$\mathbf{A}_{i,j} = \boldsymbol{k}_j^{\mathrm{T}} \left(\mathbf{D}_i^{\mathrm{T}} \boldsymbol{k}_i\right) - \left(\boldsymbol{k}_j \times \boldsymbol{p}_j\right)^{\mathrm{T}} \left(\hat{\boldsymbol{b}}_i^{\mathrm{T}} \boldsymbol{k}_i\right) = \boldsymbol{k}_j^{\mathrm{T}} \left(\mathbf{D}_i^{\mathrm{T}} \boldsymbol{k}_i\right) + \left(\boldsymbol{k}_j \times \boldsymbol{p}_j\right)^{\mathrm{T}} \left(\hat{\boldsymbol{b}}_i \boldsymbol{k}_i\right) \tag{2.53}$$

In such a way we have decoupled the dependency between indexes i and j. And we limited the cross-coupling interaction between joint j and bodies i into a minimum. The previous equation states that the effect of acceleration of each joint j is limited to the terms $\boldsymbol{k}_j^{\mathrm{T}}$, and $\boldsymbol{t}_j = \left(\boldsymbol{k}_j \times \boldsymbol{p}_j\right)$. While the effect of the articulated bodies from link n to link i is manifested by the terms $\left(\mathbf{D}_i^{\mathrm{T}} \boldsymbol{k}_i\right)$, and $\left(\hat{\boldsymbol{b}}_i^{\mathrm{T}} \boldsymbol{k}_i\right)$. The terms $\boldsymbol{d}_i = \left(\mathbf{D}_i^{\mathrm{T}} \boldsymbol{k}_i\right)$, and $\boldsymbol{y}_i = \left(\hat{\boldsymbol{b}}_i^{\mathrm{T}} \boldsymbol{k}_i\right)$ can be calculated with an $O(n)$ algorithm using multiple recursions, while the mass matrix entries can be calculated with minimum quadratic cost using the nested loop in Algorithm 2.1.

The nested loop in Algorithm 2.1. has the minimal quadratic cost. This cost results from two vector-inner products and one scalar addition, with a cost $(3n^2 + 3n)m + (2.5n^2 + 2.5n)a$, where m stands for multiplication and a stands for addition. Thus, the $O(n^2)$ computational cost is optimized.

2.5 Algorithm and variables organization in computer's memory

Figure 2.8, shows how links' models and joints' models are represented in computer's memory, while sheet i in the figure represents the dynamics of link i, the dynamics of link i is described by an acceleration of its centre of mass represented by sub-sheet $i.2$, and an inertial moment around its centre of mass represented by sub-sheet $i.1$, as such the dynamics of a link is fully described using four matrices as shown in the sub-sheets.

On the other hand sheet j is used to represent the model of joint j. The joint model is the instantaneous description of the moment acting on that joint due to the collective effect of all of the joints' accelerations and velocities. Each joint's mathematical representation is described by two matrices, one matrix is used to quantify the effects of the joints' accelerations, while the other is to describe Coriolis and centrifugal effect due to angular velocities.

The models of the joints, represented by the two matrices $\mathbf{H}_j$ and $\mathbf{G}_j$, are used eventually to calculate joint space inertia matrix and Coriolis matrix of the inverse dynamics equation in row-by-row fashion through performing projections on joint's z axis.

By considering Centrifugal accelerations only, joint space Centrifugal matrix can be calculated using the same model.

2.6 Implementation and results

To prove the validity of the proposed algorithm, acronymed (GDA), and to assess its execution-time performance, a comparison with well established algorithms was performed. Robotics Toolbox for MATLAB® (RTB) [13], CRBA algorithm by Featherstone [25] and DAMAROB toolbox for MATLAB® [36], were used for comparison. We want to mention here that computational efficiency was not the main concern of RTB, according to the author the main concern was to maintain a simple and an easy to read code. Yet after a lengthy search for robotics toolboxes, RTB according to our knowledge, is the only MATLAB® toolbox that we could find which calculates Coriolis

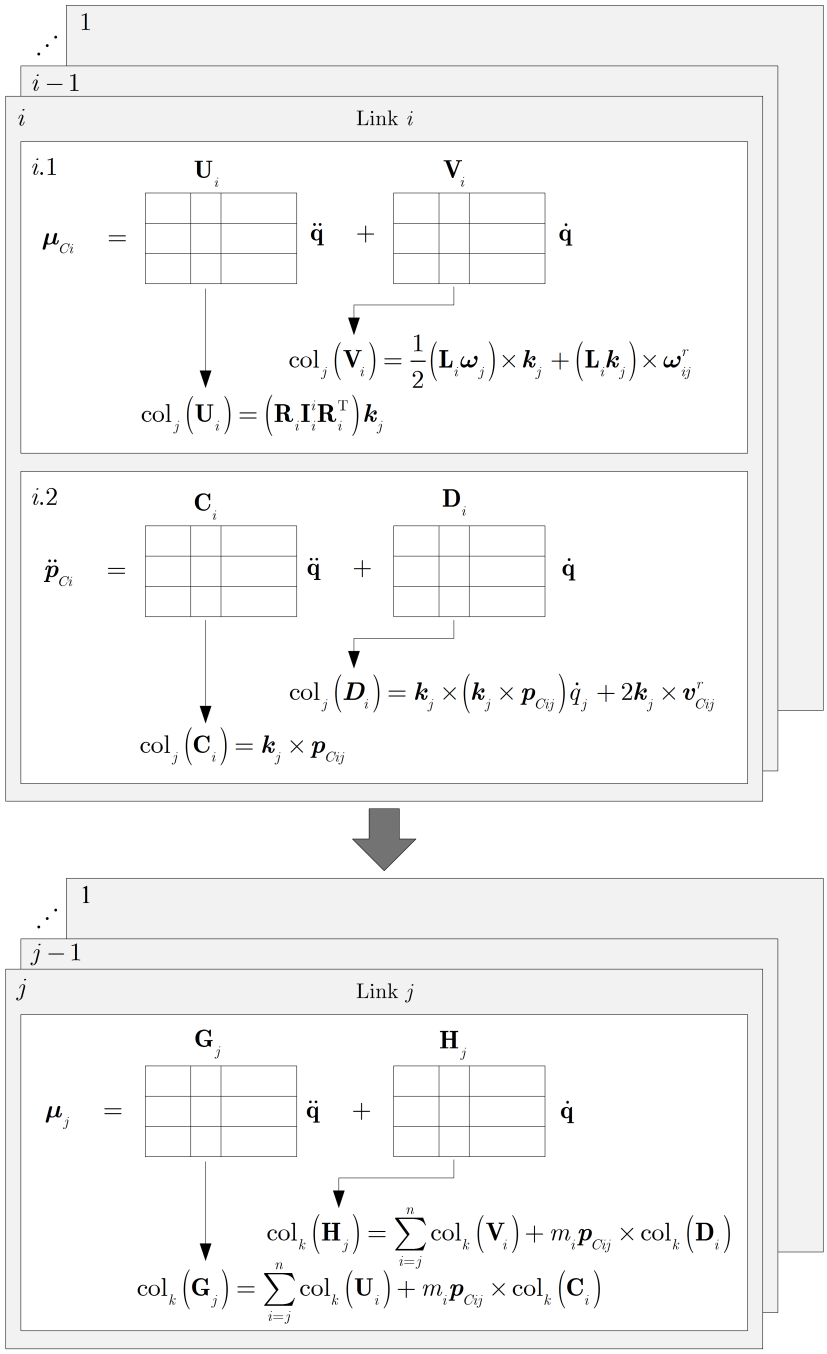

Figure 2.8: Representation of links dynamics and joints moments in memory

Table 2.1: Execution time comparison for several algorithms, applied for 6 DOF robot.

Method	Dynamic parameter	Calculation time (s)
CRBA	JSIM	0.015
RTB	JSIM	0.460
GDA	JSIM	0.021
RTB	Coriolis matrix	1.328
GDA	Coriolis matrix	0.064
GDA	Centrifugal matrix	0.030
GDA	TD-JSIM	0.069
GDA	$\mathbf{C}^{\mathrm{T}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$	0.080
Numerical differentiation	TD-JSIM	0.085
DAMAROB	JSIM	0.032

matrix numerically. DAMAROB is a Toolbox for MATLAB® that generates dynamics equations of serial manipulators in symbolic form[1].

All the code used for comparison was implemented in MATLAB®, as such six MATLAB® functions implementing the proposed algorithm were developed:

1. GetMassMatrixGDA: calculates joint space inertia matrix.
2. GetCoriolisMatrixGDA: calculates joint space Coriolis matrix..
3. GetCentrifugalMatrixGDA: calculates joint space centrifugal matrix.
4. GetDerivativeOfMassMatrixGDA: calculates the time derivative of JSIM.
5. GetCTdqGDA: calculates the time derivative of JSIM, Coriolis vector, and $\mathbf{C}^{\mathrm{T}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ as described before.
6. GetMassMatrixGDAHJ: calculates joint space inertia matrix using method proposed in 2.4, or (GDAHJ).

Table 2.1 shows the execution-time performance for the different algorithms. The profiler utility from MATLAB® was used for performing the measurements, the geometric and inertial parameters used for running the algorithms were those of PUMA 560 robotic manipulator. The tests were performed on Intel® Core™2 Duo CPU T6500 2.1 GHZ, 2 GB of RAM, working under 32b Windows® 7 operating system.

[1] Using Lagrange formulation to calculate dynamics matrices in symbolic form is a two step process, the first is performed offline where the model of the robot is provided to a symbolic-math program that automatically generates symbolic functions for calculating the dynamics matrices, those functions are only applicable for the specific robot for which they were generated. Then in the second step those functions are used online to calculate the dynamics of the robot. This is unlike the numerical calculation method, which does not require offline "preprocessing", and is not restricted to a specific robot.

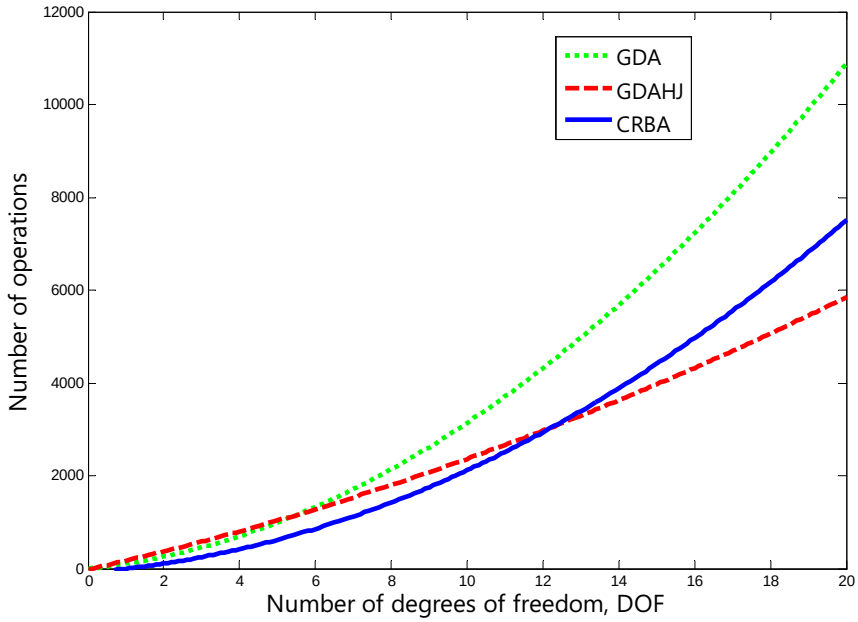

Figure 2.9: Number-of-operations with DOF, required for calculating JSIM using CRBA, GDA and GDAHJ algorithms.

From Table 2.1 we see that calculating JSIM using CRBA has the best performance, followed by GDA, then comes the symbolic method utilized in DAMAROB, then RTB in the final place.

Table 2.2 shows the computational complexity of the proposed algorithm against other algorithms, measured in the number of floating point operations, additions and multiplications, as functions of n, the number of DOF of robot. Operation count for CRBA reported in the table pertains to the most efficient version of this algorithm as reported in [27]. In the table the operation count for the various algorithms of GDA is also given.

Operation count for RTB was assessed after analysing the code. For constructing joint space inertia and Coriolis matrices, RTB invokes several calls of the efficient recursive Newton-Euler, as described in [37]. As such, RTB constructs JSIM column by column. This is done by assigning unity to only one element of joint's acceleration vector, while assigning zeros to remaining elements, joints velocities and gravity term, then recursion is performed, and the associated column vector of the inertia matrix is calculated. The same procedure is repeated for all of JSIM columns, and since that the computational complexity of recursive Newton-Euler is $O(n)$ and that constructing the inertia matrix requires n recursions, then the total computational complexity of the algorithm is of $O(n^2)$. For calculating Coriolis matrix RTB uses the same methodology it applies for calculating JSIM, that is by performing several recursions, at each recursion the angular accelerations and gravity term are set to zero, while one of the possible combinations of $(\dot{q}_i, \dot{q}_j)$ is utilized, as such RTB invokes $n^2/2$ recursions, and the total computational

complexity is of $O(n^3)$, which is reflected by the long execution time shown in Table 2.1.

From Table 2.2 we notice that for calculating JSIM, the algorithms of CRBA, GDA, and RTB are all of the same order $O(n^2)$, while the symbolic-numeric method is of order $O(n^3)$. For calculating the TDJSIM, GDA is of order $O(n^2)$, while using numerical methods results in $O(n^3)$ computational complexity. For calculating Coriolis matrix we also notice that the GDA have $O(n^2)$ complexity, while RTB and symbolic-numeric methods are of $O(n^3)$.

Figure 2.9 shows curves that represent the number of operations required to calculate JSIM as function of number of joints. The continuous thick curve represents the case when using state-of-the-art CRBA for performing the calculation. The dashed thick curve corresponds to the case when the proposed GDAHJ is used. From the curves we notice that the GDAHJ perform better for articulated bodies that possess higher degree of freedom. We want to note here that GDAHJ in its form proposed in this study, was devoted for attaining the highest efficiency possible for the $O(n^2)$ terms, nevertheless the $O(n)$ terms of the algorithm can also be optimized for attaining better overall performance, this goal will be left open for future work.

Even though CRBA performed better than GDA, it can be used only to calculate the inertia matrix. In contrast the proposed method can be used to calculate the inertia matrix, Coriolis matrix, Centrifugal matrix, and TD-JSIM matrix with computational complexity of $O(n^2)$, in addition GDAHJ achieves higher efficiency over state-of-the-art for hyper-joint manipulators. An added advantage of GDA is that when the aforementioned matrices are being evaluated robot dynamics represented by links' inertial moments and their accelerations are being calculated as by-product of the calculation. Finally we want to note that GDAHJ in its form presented in this study focused on optimizing the number of operations associated with $O(n^2)$ cost, nevertheless the $O(n)$ algorithm for calculating vectors $\boldsymbol{d}_i$ and $\boldsymbol{y}_i$ can be also optimized for achieving better efficiency. We believe that further development will render GDA and GDAHJ more efficient.

2.7 Appendix I

Almost, in all of the dynamics algorithms proposed in this study, we notice that there is always a necessity to calculate a rotation of the inertia matrix of a link represented in its local frame, which is a constant 3×3 symmetric matrix. When performed in the traditional way, this transform will cost $18 \times (3m + 2a)$, or 54 multiplications and 36 additions. In this appendix we propose an efficient method for this purpose, its computational cost is $(28m + 21a)$. To asses the performance of the proposed algorithm, a comparison with existing state-of-the-art method is presented. Our proposition compares favourably with 14% higher efficiency.

Let $\mathbf{I}$ be3×3 inertia matrix of a link, $\mathbf{R}$ is a general rotation matrix. The rotation of matrix $\mathbf{I}$ by rotation matrix $\mathbf{R}$ is given by the similarity transform $\mathbf{RIR^T}$. In the following we describe an efficient algorithm for calculating the the rotation $\mathbf{RIR^T}$. We write $\mathbf{R^T}$ as a combination of three orthonormal vectors:

Table 2.2: Operation count for proposed method and other methods, m stands for multiplication and a for addition.

Method	Matrix	Cost	Reference
GDA	JSIM	$(12n^2+49n)m+(11n^2+35n-3)a$	
GDAHJ	JSIM	$(3n^2+88n-3)m+(2.5n^2+95.5n-18)a$	
GDA	Coriolis	$(31.5n^2+53.5n-3)m+(33n^2+31n-6)a$	
GDA	Centrifugal	$(15n^2+58n)m+(13.5n^2+39.5n-3)a$	
GDA	TD-JSIM	$(46.5n^2+122.5n-36)m+(51.5n^2+117.5n-41)a$	
CRBA	JSIM	$(10n^2+22n-32)m+(6n^2+37n-43)a$	[38] and [27] EQ. 10.3
Symbolic-Numeric	JSIM-Coriolis	$(\frac{3}{2}n^3+\frac{35}{2}n^2+9n-16)m+(\frac{7}{6}n^3+\frac{23}{2}n^2+\frac{64}{3}n-28)a$	[22]
RTB	JSIM	$O(n^2)$	
RTB	Coriolis	$O(n^3)$	
Numerical differentiation	TD-JSIM	$O(n^3)$	

$$\mathbf{R^T} = [\boldsymbol{e_1}\ \boldsymbol{e_2}\ \boldsymbol{e_3}] \tag{2.54}$$

Then, we can write the transform $\mathbf{RIR^T}$, by:

$$\mathbf{RIR^T} = \begin{bmatrix} \boldsymbol{e_1^T} \\ \boldsymbol{e_2^T} \\ \boldsymbol{e_3^T} \end{bmatrix} \mathrm{I}[\boldsymbol{e_1}\ \boldsymbol{e_2}\ \boldsymbol{e_3}] \tag{2.55}$$

By performing the multiplication we find that:

$$\mathbf{RIR^T} = \begin{bmatrix} \boldsymbol{e_1^T}\mathbf{I}\boldsymbol{e_1} & \boldsymbol{e_1^T}\mathbf{I}\boldsymbol{e_2} & \boldsymbol{e_1^T}\mathbf{I}\boldsymbol{e_3} \\ \vdots & \boldsymbol{e_2^T}\mathbf{I}\boldsymbol{e_2} & \boldsymbol{e_2^T}\mathbf{I}\boldsymbol{e_3} \\ \vdots & \vdots & \boldsymbol{e_3^T}\mathbf{I}\boldsymbol{e_3} \end{bmatrix} \tag{2.56}$$

Since that $\mathbf{I}$ is symmetric, then all of its eigenvalues are real, and it can be written as a product of a diagonal $\mathbf{\Gamma}$ and orthonormal $\mathbf{Q}$ matrices:

$$\mathbf{I} = \mathbf{Q\Gamma Q^T} \tag{2.57}$$

Where $\mathbf{\Gamma}$ is a diagonal matrix of the eigenvalues of $\mathbf{I}$, or the principal moments of inertia, (B/11) of [2], and $\mathbf{Q}$ is an orthonormal matrix, which is formed by the concatenation of the eigenvectors of matrix $\mathbf{I}$, $(\boldsymbol{q_1}, \boldsymbol{q_2}, \boldsymbol{q_3})$, or the principal axes of inertia, (B/12) of [2]. Like so, $\mathbf{\Gamma}$ and $\mathbf{Q}$ are given by:

$$\mathbf{\Gamma} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \tag{2.58}$$

And

$$\mathbf{Q} = [\boldsymbol{q_1}\ \boldsymbol{q_2}\ \boldsymbol{q_3}] \tag{2.59}$$

Where $\lambda_1, \lambda_2, \lambda_3$ are the eigenvalues associated with matrix $\mathbf{I}$. We index the eigenvalues such that $\lambda_1 > \lambda_3$ and $\lambda_2 > \lambda_3$. We can rewrite $\mathbf{\Gamma}$ by:

$$\mathbf{\Gamma} = \mathbf{1_3} \cdot \lambda_3 + \begin{bmatrix} \lambda_1 - \lambda_3 & 0 & 0 \\ 0 & \lambda_2 - \lambda_3 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \mathbf{1_3} \cdot \lambda_3 + \mathbf{U} \tag{2.60}$$

Without loss of generality, the eigenvalues in matrix $\mathbf{\Gamma}$ can be in any other order, given that the inequalities $\lambda_1 > \lambda_3$ and $\lambda_2 > \lambda_3$ are held. In other words λ_3 shall be chosen such that the entries of matrix $\mathbf{U}$ are positive. By noticing that, $\mathbf{Q}$ is orthonormal, then:

$$\mathbf{Q}\mathbf{Q}^\mathbf{T} = \mathbf{1_3} \tag{2.61}$$

Then $\mathbf{I}$ can be rewritten:

$$\mathbf{I} = \mathbf{Q}\mathbf{\Gamma}\mathbf{Q}^\mathbf{T} = \mathbf{1_3}\lambda_3 + \mathbf{Q} \begin{bmatrix} \lambda_1 - \lambda_3 & 0 & 0 \\ 0 & \lambda_2 - \lambda_3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{Q}^\mathbf{T} \tag{2.62}$$

By substituting (2.62) into (2.56) and fixing, we notice that each entry (i, j) of equation (2.56) can be written as:

$$\boldsymbol{e_i}^\mathbf{T}\mathbf{I}\boldsymbol{e_j} = \delta_{ij}\lambda_3 + \boldsymbol{e_i}^\mathbf{T}\widetilde{\mathbf{Q}}\widetilde{\mathbf{Q}}^\mathbf{T}\boldsymbol{e_j} \tag{2.63}$$

Where δ_{ij} is the *Kronecker* symbol:

$$\delta_{ij} = \begin{cases} 1, & if\ i = j \\ 0, & if\ i \neq j \end{cases}$$

And $\widetilde{\mathbf{Q}}$ is given by:

$$\widetilde{\mathbf{Q}} = [\boldsymbol{q_1}\ \boldsymbol{q_2}] \begin{bmatrix} \sqrt{\lambda_1 - \lambda_3} & 0 \\ 0 & \sqrt{\lambda_2 - \lambda_3} \end{bmatrix} \tag{2.64}$$

Where $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are the eigenvectors associated with λ_1 and λ_2. Where λ_1 and λ_2 are the most and second maximum eigenvalues, and $\widetilde{\mathbf{Q}}$ is 3×2 matrix. We notice that $\widetilde{\mathbf{Q}}$ is a constant matrix, thus, it can be calculated offline, resulting in considerable reduction of the amount of computation required during the online calculations. Then we define the vector $\widetilde{\boldsymbol{s}}_\boldsymbol{i}$ of dimension 2×1 by:

$$\widetilde{\boldsymbol{s}}_\boldsymbol{i} = \widetilde{\mathbf{Q}}^\mathbf{T}\boldsymbol{e_i}, \qquad i = 1, 2, 3 \tag{2.65}$$

In such a way, calculating all vectors $\widetilde{\boldsymbol{s_i}}$ requires $3 \times (6m + 4a)$. And the matrix $\mathbf{RIR^T}$ can be rewritten as:

$$\mathbf{W} = \mathbf{RIR^T} = \begin{bmatrix} \lambda_3 + \widetilde{\boldsymbol{s_1}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\mathbf{1}} & \widetilde{\boldsymbol{s_1}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\mathbf{2}} & \widetilde{\boldsymbol{s_1}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\mathbf{3}} \\ \vdots & \lambda_3 + \widetilde{\boldsymbol{s_2}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\mathbf{2}} & \widetilde{\boldsymbol{s_2}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\mathbf{3}} \\ \vdots & \vdots & \lambda_3 + \widetilde{\boldsymbol{s_3}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\mathbf{3}} \end{bmatrix} \tag{2.66}$$

Calculating the previous matrix from $\tilde{\boldsymbol{s}}_{\boldsymbol{i}}$ and λ_3 requires: $6 \times (2m + 1a)$ operations associated with the terms $\widetilde{\boldsymbol{s}}_{\boldsymbol{i}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\boldsymbol{j}}$, and three additions for adding λ_3 in the diagonal.
The aforementioned algorithm can be enhanced further by noticing that the trace of a tensor is invariant under rotation. Then we can calculate w_{33} entry of the rotated tensor from the other two diagonal entries and the trace from:

$$w_{33} = \lambda_3 + \widetilde{\boldsymbol{s}}_{\mathbf{3}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\mathbf{3}} = trace(\mathbf{I}) - w_{22} - w_{33} \tag{2.67}$$

By noticing that $trace(\mathbf{I})$ is a constant, thus it can be calculated offline, exploiting this property, reduces the amount of computation further by 2 multiplications, while keeping the number of additions unchanged, as a result the total number of operations required is:

$$28m + 21a$$

The algorithm, accompanied with break down of its cost, is shown in Algorithm 2.2.

Algorithm 2.2 The proposed algorithm, and its cost, for efficient computation of the rotation of a symmetric matarix

Operation:	Cost:
Calculate vectors $\widetilde{\boldsymbol{s}}_{\boldsymbol{i}} = \widetilde{\mathbf{Q}}^{\mathbf{T}} \boldsymbol{e_i},\ i = 1, 2, 3:$	$3 \times (6m + 4a)$
Calculate products $\widetilde{\boldsymbol{s}}_{\boldsymbol{i}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\boldsymbol{j}},\ i = 1, 2/j = 1, 2, 3:$	$5 \times (2m + 1a)$
Add λ_3 to the terms $\widetilde{\boldsymbol{s}}_{\boldsymbol{i}}^{\mathbf{T}} \tilde{\boldsymbol{s}}_{\boldsymbol{i}},\ i = 1, 2:$	$2 \times (1a)$
Calculate w_{33} from $w_{33} = trace(\mathbf{I}) - w_{22} - w_{33}$	$2 \times (1a)$
Total number of operations:	$28m + 21a$

Comparison

In appendix A.5, of [27], Featherstone presented an efficient algorithm for computing the rotation of a symmetric matrix $\mathbf{I}$, which was accomplished by the following decomposition of matrix $\mathbf{I}$:

$$\mathbf{I} = \mathbf{L} + \mathbf{D} + \boldsymbol{v}\times \tag{2.68}$$

Where $\boldsymbol{v}\times$ is the skew symmetric matrix associated with vector $\boldsymbol{v}$. And matrices $\mathbf{L}$ and $\mathbf{D}$, and vector $\boldsymbol{\nu}$ are given by:

$$\mathbf{L} = \begin{bmatrix} I_{11} - I_{33} & I_{12} & 0 \\ I_{12} & I_{22} - I_{33} & 0 \\ I_{31} + I_{13} & I_{32} + I_{23} & 0 \end{bmatrix}$$

$$\mathbf{D} = I_{33} \cdot \mathbf{1_3}$$

$$\boldsymbol{v} = \begin{bmatrix} -I_{23} \\ I_{13} \\ 0 \end{bmatrix}$$

Where I_{ij} are the (i, j) entries of matrix $\mathbf{I}$, nevertheless, the resulting cost of the presented algorithm is $(28m + 29a)$, which requires in total 57 operations. In other words the method of [27] requires 8 extra operations over our proposition.

In this study, we proposed an efficient algorithm for computing the rotation transform of 3×3 symmetric matrices. We propose to store any 3×3 symmetric matrix $\mathbf{I}$, in a data structure of the form$(\lambda_3, t, \widetilde{\mathbf{Q}}^{\mathbf{T}})$, where λ_3 is he minimum eigenvalue, t is the trace of the inertia tensor, and $\widetilde{\mathbf{Q}}^{\mathbf{T}}$ is (2×3) matrix given in (2.64). Using our proposition the total number of operations required to calculate the rotation transform is reduced from $(54m+36a)$ using the conventional way, to the more efficient operation count$(28m+21a)$, as illustrated in Algorithm (2.2). The proposed algorithm compares favourably with state-of-the-art method, and delivers 14% better performance.

2.8 Appendix II

In the following we present a prove of frame-injection principal that we adopted throughout this paper. Let $\mathbf{P}$ be a point of interest of link i, let $\boldsymbol{p}_i^0$ be the position vector of this point in the base frame, and $\boldsymbol{p}_i^i$ the position vector of this point represented in frame i, as shown in Figure 2.10, and let $\mathbf{T}_k^{k-1}$ be the 4×4 transformation matrix from frame k to frame $k-1$ then the relationship between $\boldsymbol{p}_i^0$ and $\boldsymbol{p}_i^i$ is given by:

$$\boldsymbol{p}_i^0 = \mathbf{T}_1^0 \mathbf{T}_2^1 \mathbf{T}_3^2 \dots \mathbf{T}_i^{i-1} \boldsymbol{p}_i^i \tag{2.69}$$

For deriving an expression of the acceleration of point $\mathbf{P}$ we take the second time derivative of the previous equation while taking into consideration that the vector $\boldsymbol{p}_i^i$ is constant, as such $\ddot{\boldsymbol{p}}_i^0$ can be written as follows:

$$\ddot{\boldsymbol{p}}_i^0 = \boldsymbol{e}_1 + \boldsymbol{e}_2 + \boldsymbol{e}_3 + \dots + \boldsymbol{e}_i \tag{2.70}$$

While expressions of vectors $\boldsymbol{e}_k$ are given in Table 2.3, the expressions of $\boldsymbol{e}_k$ involve the transformation matrices and their derivatives, and since that the transformation matrix $\mathbf{T}_k^{k-1}$ is given by:

$$\mathbf{T}_k^{k-1} = \begin{bmatrix} \mathbf{R}_k^{k-1} & \boldsymbol{p}_k^{k-1} \\ 0 & 1 \end{bmatrix} \tag{2.71}$$

Where $\mathbf{R}_k^{k-1}$ is the rotation matrix from frame k to frame $k-1$, and $\boldsymbol{p}_k^{k-1}$ is the coordinate vector of origin of frame k described in frame $k-1$.

By taking the first derivative of the aforementioned matrix, and realizing that $\boldsymbol{p}_k^{k-1}$ is a constant vector according to MDH convention, and that $\dot{\mathbf{R}}_k^{k-1} = \mathbf{S}(\boldsymbol{\omega}_k)\mathbf{R}_k^{k-1}$ for

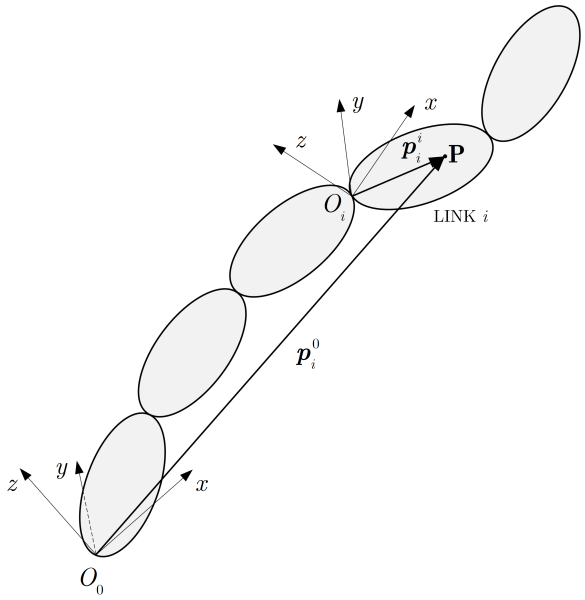

Figure 2.10: Coordinate vectors of point of interest $\mathbf{P}$ on link i of a serial robot.

Table 2.3: Acceleration terms of a point of interest P from a link i, described in reference frame.

Term	Equivalent expression
$\boldsymbol{e}_1$	$\ddot{\mathbf{T}}_1^0\mathbf{T}_2^1\mathbf{T}_3^2\dots\mathbf{T}_i^{i-1}\boldsymbol{p}_i^i + 2\dot{\mathbf{T}}_1^0\dot{\mathbf{T}}_2^1\mathbf{T}_3^2\dots\mathbf{T}_i^{i-1}\boldsymbol{p}_i^i + 2\dot{\mathbf{T}}_1^0\mathbf{T}_2^1\dot{\mathbf{T}}_3^2\dots\mathbf{T}_i^{i-1}\boldsymbol{p}_i^i + \cdots\cdots$ $+2\dot{\mathbf{T}}_1^0\mathbf{T}_2^1\mathbf{T}_3^2\dots\dot{\mathbf{T}}_i^{i-1}\boldsymbol{p}_i^i$
$\boldsymbol{e}_2$	$\mathbf{T}_1^0\ddot{\mathbf{T}}_2^1\mathbf{T}_3^2\dots\mathbf{T}_i^{i-1}\boldsymbol{p}_i^i + 2\mathbf{T}_1^0\dot{\mathbf{T}}_2^1\dot{\mathbf{T}}_3^2\dots\mathbf{T}_i^{i-1}\boldsymbol{p}_i^i + \cdots\cdots + 2\mathbf{T}_1^0\dot{\mathbf{T}}_2^1\mathbf{T}_3^2\dots\dot{\mathbf{T}}_i^{i-1}\boldsymbol{p}_i^i$
$\boldsymbol{e}_3$	$\mathbf{T}_1^0\mathbf{T}_2^1\ddot{\mathbf{T}}_3^2\dots\mathbf{T}_i^{i-1}\boldsymbol{p}_i^i + \cdots\cdots + 2\mathbf{T}_1^0\mathbf{T}_2^1\dot{\mathbf{T}}_3^2\dots\dot{\mathbf{T}}_i^{i-1}\boldsymbol{p}_i^i$
$\vdots$	$\vdots$
$\boldsymbol{e}_i$	$\mathbf{T}_1^0\mathbf{T}_2^1\mathbf{T}_3^2\dots\ddot{\mathbf{T}}_i^{i-1}\boldsymbol{p}_i^i$

a revolute joint, while $\mathbf{S}(\boldsymbol{\omega}_k)$ is a skew symmetric matrix of the angular velocity, then the first time-derivative of the transformation matrix becomes:

$$\dot{\mathbf{T}}_k^{k-1} = \begin{bmatrix} \mathbf{S}(\boldsymbol{\omega}_k)\mathbf{R}_k^{k-1} & 0 \\ 0 & 0 \end{bmatrix} \tag{2.72}$$

Also calculating the second time-derivative of the transformation matrix is straightforward:

$$\ddot{\mathbf{T}}_k^{k-1} = \begin{bmatrix} \dot{\mathbf{S}}(\boldsymbol{\omega}_k)\mathbf{R}_k^{k-1} + \mathbf{S}^2(\boldsymbol{\omega}_k)\mathbf{R}_k^{k-1} & 0 \\ 0 & 0 \end{bmatrix} \tag{2.73}$$

If we substituted the first and the second time derivatives of the transformation matrices into equations of vectors $\boldsymbol{e}_1$ up to $\boldsymbol{e}_i$ of Table 2.3, we can reinterpret the equations as follows:

- The equation of $\boldsymbol{e}_1$ can be interpreted as the injection of frame 1 into point $\mathbf{P}$ of link i in terms of a tangential and centrifugal accelerations represented by the first-most term of the equation, in-addition to an injection of Coriolis acceleration represented by the rest of the terms of the same equation.
- The equation of $\boldsymbol{e}_2$ represents the injection of frame 2 into point $\mathbf{P}$ in terms of tangential and centrifugal accelerations, also represented by the first term of that equation, while the remaining terms represent the injection of Coriolis acceleration.
- The same applies on the equation of $\boldsymbol{e}_3$ and the following equations up to $\boldsymbol{e}_i$ which represents the injection of frame i into point $\mathbf{P}$ in terms of tangential and centrifugal accelerations, while the effect of Coriolis acceleration is zero due to the fact that the velocity of any point of link i relative to frame i is zero.

Thus, if we re-notate $\boldsymbol{e}_1$ by $\ddot{\boldsymbol{p}}_{i1}$, $\boldsymbol{e}_2$ by $\ddot{\boldsymbol{p}}_{i2}$ and $\boldsymbol{e}_i$ by $\ddot{\boldsymbol{p}}_{ii}$ then we can reformulate the total acceleration of point $\mathbf{P}$ by:

$$\ddot{\boldsymbol{p}}_i = \sum_{j=1}^{i} \ddot{\boldsymbol{p}}_{ij} \tag{2.74}$$

Where $\ddot{\boldsymbol{p}}_{ij}$ is the acceleration of point $\mathbf{P}$ of link i due to injection of frame j, which is a combination of tangential $\ddot{\boldsymbol{p}}_{ij}^{\tau}$, normal $\ddot{\boldsymbol{p}}_{ij}^{n}$ and Coriolis $\ddot{\boldsymbol{p}}_{ij}^{cor}$ accelerations as in:

$$\ddot{\boldsymbol{p}}_{ij} = \ddot{\boldsymbol{p}}_{ij}^{\tau} + \ddot{\boldsymbol{p}}_{ij}^{n} + \ddot{\boldsymbol{p}}_{ij}^{cor} \tag{2.75}$$

Where $\ddot{\boldsymbol{p}}_{ij}^{\tau}$ is:

$$\ddot{\boldsymbol{p}}_{ij}^{\tau} = \boldsymbol{\varepsilon}_j \times \boldsymbol{p}_{ij} \tag{2.76}$$

Where $\boldsymbol{p}_{ij}$ is the vector connecting the origin of frame j to point $\mathbf{P}$. Also $\ddot{\boldsymbol{p}}_{ij}^{n}$ is given by:

$$\ddot{\boldsymbol{p}}_{ij}^{n} = \boldsymbol{\omega}_j \times (\boldsymbol{\omega}_j \times \boldsymbol{p}_{ij}) \tag{2.77}$$

And $\ddot{\boldsymbol{p}}_{ij}^{cor}$ is:

$$\ddot{\boldsymbol{p}}_{ij}^{cor} = 2\boldsymbol{\omega}_j \times \boldsymbol{v}_{ij}^{r} \tag{2.78}$$

Where $\boldsymbol{v}_{ij}^{r}$ is similar to $\boldsymbol{v}_{Cij}^{r}$ described in equation (2.10), but it differs in that it describes the velocity of the point of interest $\mathbf{P}$ rather than the centre of mass of link i.

The total acceleration of the point of interest had been derived. In case point $\mathbf{P}$ is chosen to be coincident with the centre of mass of link i then the equation of total acceleration of point $\mathbf{P}$ (2.74) renders back to equation (2.18) representing the acceleration of the centre of mass of link i.

Deducing the equation of inertial moment $\boldsymbol{\mu}_{\mathrm{C}i}$ of link i, used in this paper, is easily done by integrating the inertial moments of all of the particles of link i throughout the volume of that link.

Chapter 3

COLLISION AVOIDANCE

3.1 State of the art

Collision avoidance is a major topic in robotics research, especially in the new context of collaborative robotics. Owing to its importance, numerous studies approaching collision avoidance had been presented. Nevertheless, we can notice that the variety of methods proposed, though different in their own right, have some underlying similarities and can be divided into four major classes. Based on the principle on which those methods are built, those classes are:

1. Potential fields (PF) principal;
2. Optimization techniques;
3. Heuristic methods;
4. Others.

3.1.1 Potential fields

In PF-based methods the robot is in a hypothetical vector field influenced by two types of forces. Forces of attraction that guide the robot towards the goal, and repulsion forces that repel it away from obstacles. Subjected to these forces the robot finds its way to the goal while avoiding collisions.

One of the most revolutionary methods proposed in this category was the artificial potential field [39]. Unlike its predecessors, this method was the first of its kind that treated the problem of collision avoidance at the low-level control, which is best suited for achieving real-time response. Owing to its simplicity and high performance this method was used as the bases for this study.

Another method that is based on a similar principal is the 3 Dimensional (3D) force field method [40]. In this method each link of the robot is surrounded by a virtual elliptical volume. When the obstacle penetrates into the ellipsoid, a hypothetical repulsive force is generated, repelling the robot away from it and avoiding collision.

Based on the PF principal, an approach to collision avoidance and trajectory planning was proposed in [41]. This method requires as an input a preliminary trajectory,

generated by other method. Repulsion poles are used to represent obstacles. An attraction pole is utilized for guiding the robot toward the goal. The attraction pole moves along the preliminary trajectory and the robot will follow it while being repelled away from obstacles. This method is suitable for generating collision free paths that are as close as possible to a predefined trajectory. A similar implementation is adopted in this study.

Recently, in [42], a depth space approach for collision avoidance between a robot and coworker was presented. The study describes an improved implementation of the artificial potential field method in which an estimation of obstacle's velocity was taken into consideration when computing the repulsion vector. This is an advancement to the original artificial potential field that utilizes only information about minimum distances for calculating the repulsion vectors. Also, the paper proposed a novel approach for estimating obstacle's velocity.

While the artificial potential field is inspired by electric field phenomena, other approaches, inspired by other types of fields, were proposed. The circular fields method, inspired by electromagnetism, was investigated in [43]. One of the advantages offered by this method over the artificial potential field is that it is immune to getting stuck in local minima, a major drawback in the artificial potential field.

Other researchers drew inspiration from fluid dynamics phenomena. In this context, [44] utilized the stream lines of potential flow for attaining collision free paths. The Circular Theorem from fluid dynamics was implemented for computing the stream lines of the potential flow. Similarly, in [45] collision free paths are attained by superimposing elementary flows. Doublets are used to model cylindrical obstacles inserted in a uniform flow. In such a way, collision free paths are attained after calculating the stream lines of the flow. Analogous to circular fields, the methods based on fluid dynamics are also immune to dead-lock (getting stuck in local minima). Nevertheless, almost all of the studies based on fluid mechanics techniques are dedicated to applications of collision avoidance for mobile robots.

In [46] the authors proposed the application of biharmonic potential functions for collision free path calculation, inspired by the phenomena of stress distribution in materials. In this method, the navigation zone is considered to be a continuous solid with elasticity properties, obstacles are voids inside the solid, and the goal is modelled by a pressurized cavity. The distribution of stresses inside the material is calculated and from this distribution a collision free path is established.

3.1.2 Optimization techniques

Apart from the methods based on the potential fields, several researchers have approached robot collision avoidance as an optimization problem. In [2] the authors utilized the square of a norm of an error vector as objective function. This error was defined from the difference between the end-effector's velocity and the desired velocity. The constraints are formulated from limits on (1) joints angles, (2) joints velocities and (3) the likelihood of collision. The optimization problem is solved in real-time using the logarithmic barrier method, and the computed velocity is used to command the robot. To test their algorithm they used two different setups: (1) using two robotic manipulators, Figure 3.1, and (2) a robot avoiding collision with a spherical obstacle, Figure

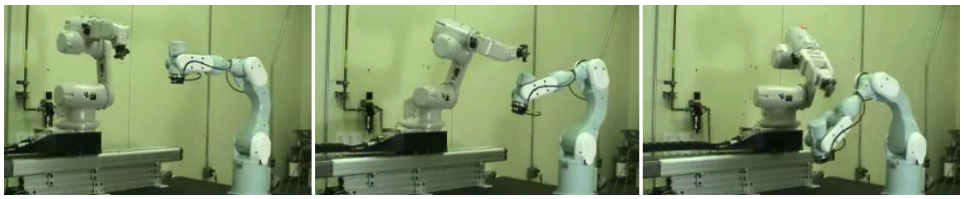

Figure 3.1: Collision avoidance between two robots [2]

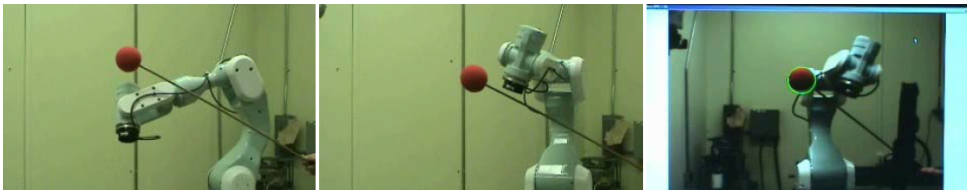

Figure 3.2: Collision avoidance between a spherical obstacle and a manipulator [2]

3.2. In the first setup, on of the manipulators is mounted on a prismatic base and is commanded to move on a preprogrammed trajectory while the collision avoidance algorithm is used to command the second robot. In the second setup, the manipulator has to avoid collision with a spherical obstacle moving in the scene.

Another motion planner that takes into consideration obstacle avoidance is presented in [47]. This planner employs both the potential field and a genetic optimization algorithm. In this method the planning process is divided into two procedures. In the first the artificial potential fields method was employed to find a collision free path for the end-effector. In the second procedure, collision free configurations were calculated using Genetic Optimization techniques. The method was validated by performing simulations on a virtual robot with twelve DOF.

In [48] the authors described an algorithm for collision avoidance for redundant manipulators. The algorithm operates at the kinematics (joint-velocities) level. At first, the minimum distance between the robot and the obstacle is calculated, if closer than a predefined safety distance, a motion component is assigned to the part of the robot closest to the obstacle, repelling the robot away from collision. The novelty in their approach was in the way the repulsion motion was defined, or what they call "one dimensional operational space" approach. In such approach, the repulsion component is projected on a certain direction aligned with the closest distance between the robot and the obstacle. This way of defining the repulsion motion gave their algorithm better immunity against singularities. Nonetheless, the proposed method was built around the fact that the end-effector's task is the primary task, or the task with the highest-level priority, while collision avoidance was treated in the null space of the end-effector's Jacobean. This will cause the manipulator to fail in avoiding collision with the obstacle in some situations. For example, when there is a contradiction between the end-effector's task, assigned the highest-level priority, and the collision avoidance task, assigned a lower-level of priority. For solving this drawback the authors proposed to re-invoke a high-level path planning algorithm, so that a new collision free path can be calculated. Another solution for this problem, more efficient and not requiring re-

planning, is proposed in [49]. In this solution a coefficient is introduced into the control equation, and by changing the value of this coefficient the priority level between tasks can be switched smoothly.

In [50] the authors proposed an inverse kinematics solver that takes into consideration joint limits and collision avoidance. The inverse kinematics problem was formulated as a minimization of the square of the error between the goal position and the end-effector position. The minimization is subjected to (1) constraints due to joints limits, and (2) constraints due to restrictions on minimum distance between the obstacle and the robot. Fiacco and McCormick algorithm was used to solve the optimization problem. The method was tested on a 5 DOF robotic manipulator.

3.1.3 Heuristic methods

The most famous method of this category is the Probabilistic Road Maps (PRM), [51]. PRM is a powerful method for generating collision free paths for robots with high degrees of freedom in non-dynamic environments where obstacles are stationary in space. This method is composed of two phases. The first phase is called the learning phase, it is performed offline, and used for stochastic generation of a *road map* of the scene. The second phase is called the query phase. In this phase, paths are queried, and the *road map* is searched for a feasible path. While PRM is a powerful method, it suffers two major problems. The first of which is that it is unsuitable for real-time implementation. This is due to the high computational cost of the method, particularly for robots with high degrees of freedom. The second drawback comes from the fact that the resulting trajectories generated by this planner are not dynamically optimized for direct implementation on the robot. So much so, PRM is more suited for the offline global planning over real-time collision avoidance implementation. Nevertheless, due to the advancement in processing power of contemporary computers, and despite the high computational cost of PRM, a recent study [52] reported success in implementing PRM for real-time collision free path planning in dynamic environments. Based on PRM, in [53] the authors proposed the dynamic road maps, a high level algorithm for planning dynamic paths in changing environment, and more suitable for real-time collision avoidance applications. This method was used successfully in [54], where the authors presented a collaborative human-robot system with integrated collision avoidance capability.

In [55] the authors presented a collision avoidance system in which the control strategy proposed searches for a motion-direction of the end-effector that guarantees a collision free path between the robot and the coworker. In case this direction exists, the end-effector is commanded to move in that direction. In case the direction does not exist the robot is stopped, while waiting for the coworker to move from its way.

3.1.4 Others

Another body of research has taken a simpler approach for collision avoidance. The tradeoff for this simplicity is manifested by (1) a limited reaction capabilities of the robot and/or (2) a requirement for a complete knowledge of object's trajectory. Though restrictive, the methods that require a total knowledge of object's trajectory are ap-

pealing when performing collision avoidance between different robotic manipulators. For example, in multi-robot cells or for dual-arm manipulators, where the trajectories of the manipulators are known a priori. A study that utilizes this knowledge for achieving collision avoidance between two end-effectors is presented in [56]. In this study the end-effectors are modelled as spheres, using their pre-calculated trajectories, a collision map is constructed. Afterwards, an iterative time shifting algorithm is applied and the time delay required to achieve collision avoidance is calculated. In other words, collision free operation is achieved by performing time-rescheduling on motion commands before sending them to controllers. This approach was developed further in [57], where an Advanced Collision Map is introduced, the method can be applied for collision avoidance between two manipulators rather than their end-effectors only. In [58] the method was generalized to perform collision avoidance between any number of manipulators. In short, this method is powerful and oriented for collision avoidance between manipulators sharing the same workspace.

A study dedicated for collision avoidance between two manipulators for industrial applications is in [59]. The problem was addressed in a simple way, by dividing the work space of the manipulators into a shared work area, accessible to both manipulators, and an external work area accessible to only one manipulator. The authors added a processing layer into the control structure, in which point to point control commands are processed before being sent to the controllers. As consequence, the manipulators are allowed to operate in their own external work area at any time. However, the presence of one of the manipulators inside the shared work area will deny access to the other manipulator, causing it to wait until the shared work area is free from the other manipulator.

Another simplified approach for collision avoidance between a manipulator and obstacles is to stop the manipulator as quickly as possible when the minimum distance between the manipulator and obstacle is smaller than a predefined safety distance. This approach is addressed in [60].

All of the studies previously described focused on collision avoidance between remote obstacles and the robot. However, in the case of redundant manipulators, self-induced collisions are also possible. A Representation of Body by Elastic Elements (RoBE) is a method used for avoiding robot self-collisions, in which each link is covered by a fictitious elastic elements [61]. Whenever the elements touch, a force is generated, and collision avoidance is achieved.

3.2 Proposed approach

In concept, the proposed strategy for achieving collision avoidance is similar to the one used in [62]. Though, unlike [62], we apply the method in the context of real-time collision avoidance, rather than the offline path planning problem. The control objective will be divided into two separate tasks. The purpose of the first task is to generate a motion that satisfies a predefined trajectory of the end-effector, while the purpose of the second task is to generate collision avoidance motion. In such a way, the resulting control signal will be the superposition of the two motions. This superposition can be performed either in joint space as in [62], or more efficiently in

operational space as proposed in this study.

3.3 Implementation overview

For providing a manipulator with collision avoidance capability several tasks had to be implemented. This section can be considered as an outline to the discussion of the next sections, where an in-depth overview, accompanied with mathematical treatment of each topic is provided. As we mentioned before, the artificial potential field was chosen as the bases for our study. The obstacles and the robot are represented by geometrical primitives, after the geometrical representation, a computationally efficient method for calculating the minimum distance between the robot and obstacles is implemented. This method has the capability of computing the coordinates of the minimum distance points from the obstacle and the robot. Then, we present the conventional ways for calculating the repulsion forces, based on the information attained in the minimum distance calculation and the methodology for computing the attraction forces that attract the end-effector to the goal is presented. Afterwards, robot's kinematics and dynamics is implemented, and the controller is established. Two types of controllers are implemented, one is based on kinematics and the other is a force-based controller. Finally, additional functionalities had been added: (1) work space limits avoidance, (2) joints limits avoidance and (3) self-collision avoidance. To validate the algorithms proposed in this study, real-time virtual-reality simulations were carried out. Two programs were developed. One is a MATLAB® source code for numerical calculations and is used to run the collision avoidance algorithm. The other program was implemented for V-REP which is mainly used for visualization. The communication between the two programs was established through sockets, Figure 3.3.

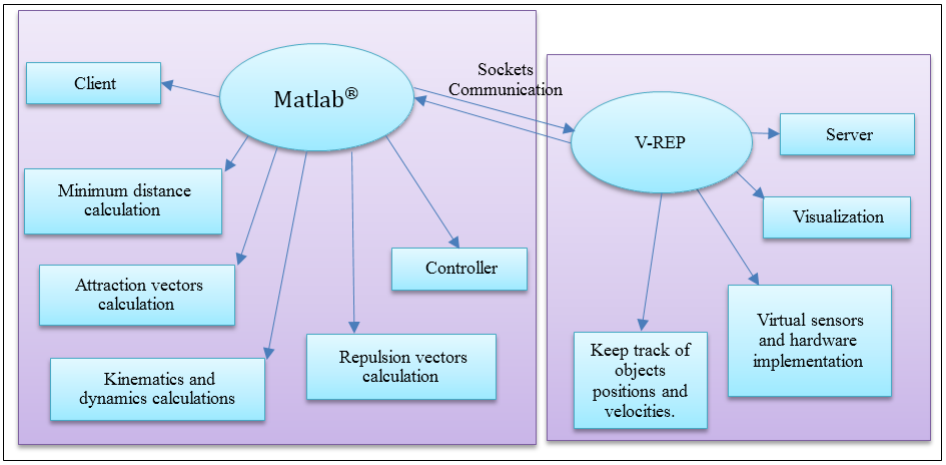

Figure 3.3: The main building blocks for collision avoidance

3.4 Potential field method

The artificial potential field method introduced by the pioneering work of Khatib [39] is one of the most famous methods for performing real-time collision avoidance. This method was applied successfully for both manipulators and mobile robots. In this method the robot is subjected to two types of potential fields. One represents the attraction forces that pull the robot toward the goal position, while the other represents the repulsion forces that push the robot away from obstacles in the environment. Those forces are defined by the gradient of the potential field. As a result the robot will move toward the goal while avoiding collisions with obstacles. Owing to its additive property, the total potential field is the sum of the attraction and the repulsion potentials:

$$U_{total} = U_{att} + U_{rep} \tag{3.1}$$

Figure 3.4 demonstrates the artificial potential fields concept and illustrates their additive property. The total field is the sum of the attraction field, with forces pulling towards the goal, and the repulsion field, with forces pointing away from obstacle. Owing to its computational efficiency the method can be implemented in real-time, and can be deployed easily into the low-level control system of the robot. The use of this method had been extended successfully to offline global path planning problems. However, in the context of this study we limit the discussion to real-time collision avoidance implementation.

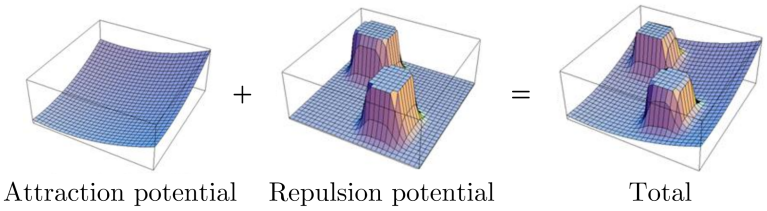

Figure 3.4: Artificial potential field concept.

3.5 Geometrical representation

To implement a collision avoidance algorithm it is of vital importance to have a geometrical representation of the robot and the obstacles. In the following sections, an overview of some existing methods for geometrical representations of the robot and the obstacles are presented. Based on the type of representation chosen, a computationally efficient algorithm to calculate the minimum distance between obstacle and the robot is proposed.

3.5.1 Representing objects geometry

Several methods had been proposed in literature for geometrical representation of the robot and the obstacles. A method of representation utilizes convex polyhedrons, this method is used in [63] to represent two PUMA 560 manipulators. In [64] the authors opted for ellipses to represent the links of the robot, while obstacles were represented by spheres. Another technique that is computationally efficient represents the robot and the obstacles by primitive shapes, as segments of lines with spheres swept onto them, [2] and [62]. A similar convention was implemented in [65], where obstacles and robot are represented by spheres and cylinders. Also, in [66] the authors represented a humanoid robot by cylindrical shapes for efficient implementation of self collision avoidance algorithm. For efficient calculation of the minimum distance, in [55] the authors opted to represent the robot and the coworker by a collection of spheres.

Another technique, which is more precise for representing the robot and the environment around it, utilizes mesh representation [1]. This type of representation has the disadvantage of being extremely costly in terms of computations for performing minimum distance calculations between the robot and the obstacle. To speed up the computation, researchers have utilized the power of parallel processing, and GPU processors, to carry out the calculations [1]. Nevertheless, programming a GPU is not an easy task, it requires considerable amount of time and special skills, so it had been avoided for the sake of this study. Thus, the primitive geometry method was our choice for this study.

3.5.2 Obstacles representation

Obstacles are represented by primitive shapes, spheres and cylinders. The spherical primitive is defined by the coordinates of its center in addition to its radius. The cylindrical primitive is defined by a line segment at its axis in addition to its radius. In this study the cylinder representation is used for the coworker. For a more precise representation of obstacles, a collection of primitive shapes, several cylinders or mixture of cylinders and spheres, can be used together.

3.5.3 Robot representation

To perform fast calculation of the minimum distance between the robot and obstacles, the robot's geometry is simplified. Two cylinders were used to represent it, one cylinder runs through the upper arm of the manipulator and another runs through the lower arm, Figure 3.5.

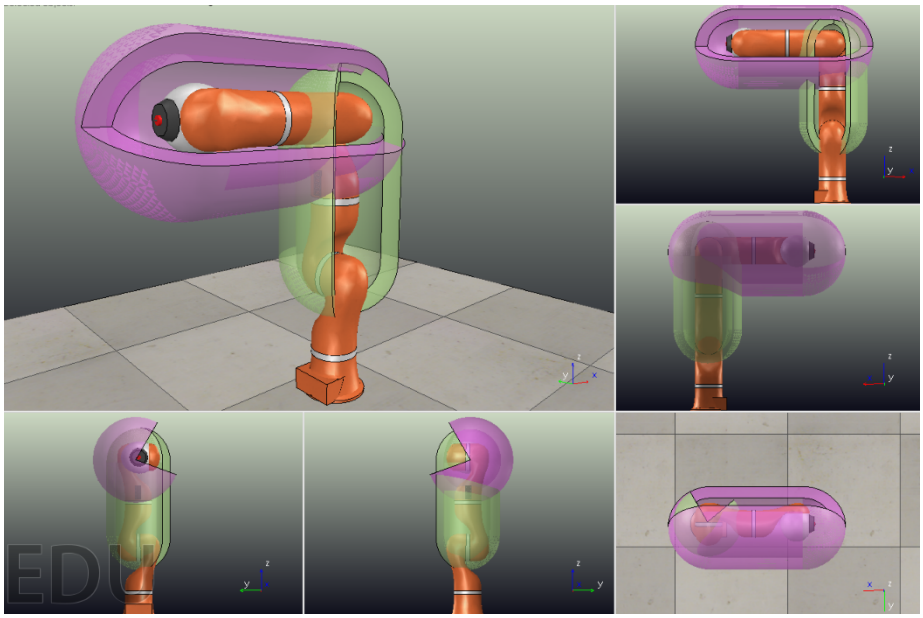

Figure 3.5: A robot represented by cylindrical primitives superimposed on it

3.6 Minimum distance calculation

A methodology for calculating the minimum distance between the robot and obstacle(s) is described. Since that we are using a simplified representation of the robot and the obstacle as primitive shapes, we describe the methodology for calculating the minimum distance analytically. This is because a closed form analytic solution can be deduced with minimum effort, it is computationally efficient and achieves real-time performance.

According to the type of objects between which the minimum distance is to be calculated, we can distinguish three different cases: (1) the two primitives are spheres, (2) one of the primitives is a cylinder while the other is a sphere, (3) the two primitives are cylindrical primitives.

3.6.1 Minimum distance between two spheres

The minimum distance is the center-distance between the two spheres minus the sum of the radiuses of the two spheres. As shown in Figure 3.6, $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ are the position vectors of the center of the two spheres, and ρ_1 and ρ_2 are the radiuses of the first and the second spheres, respectively. The minimum distance can be calculated from the relation:

$$d_{min} = |\boldsymbol{p_2} - \boldsymbol{p_1}| - \rho_1 - \rho_2 \tag{3.2}$$

3.6.2 Minimum distance between a cylinder and a sphere

This case represents the situation when calculating the minimum distance between the robot (cylinder) and an obstacle (sphere), Figure 3.7. The sphere's center, in relation

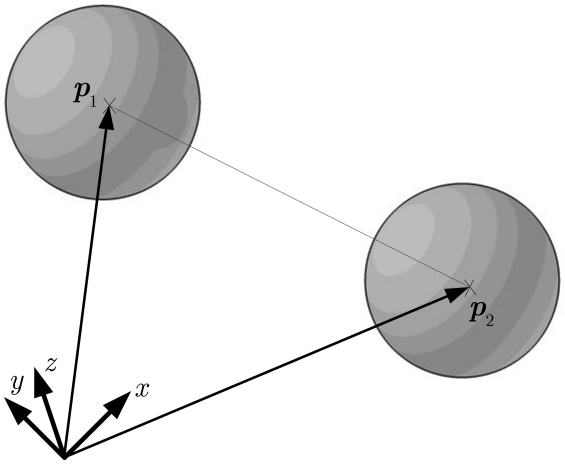

Figure 3.6: Minimum distance between two spheres

to base frame, is given by the vector $\boldsymbol{p_0}$, while the cylinder's primitive is the segment line defined by the vectors $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$. Vector $\boldsymbol{u}$ represent the point associated with the projection of the sphere's center on the line segment of the cylinder. According to the position of the obstacle's center $\boldsymbol{p_0}$ relative to the robot's primitive-segment $\boldsymbol{L}$, $\boldsymbol{p_0}$ and $\boldsymbol{L}$ can be in one of three different configurations.

3.6.2.1 First configuration

The projection of point $\boldsymbol{p_0}$ on the primitive segment $\boldsymbol{L}$ is in between its two ends $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$. To identify this case the following condition shall be satisfied:

$$(\boldsymbol{p_0} - \boldsymbol{p_1})^{\mathrm{T}}(\boldsymbol{p_2} - \boldsymbol{p_1}) > 0 \quad \textit{and} \quad (\boldsymbol{p_0} - \boldsymbol{p_2})^{\mathrm{T}}(\boldsymbol{p_1} - \boldsymbol{p_2}) > 0 \tag{3.3}$$

The coordinate's vector $\boldsymbol{u}$ of the point from the primitive associated with the minimum distance is calculated from:

$$\boldsymbol{u} = \boldsymbol{p_1} + (\boldsymbol{p_0} - \boldsymbol{p_1})^{\mathrm{T}} \frac{(\boldsymbol{p_2} - \boldsymbol{p_1})}{|\boldsymbol{p_2} - \boldsymbol{p_1}|} \tag{3.4}$$

While the associated minimum distance is given by:

$$d_{min} = |\mathbf{u} - \mathbf{p_0}| - \rho_1 - \rho_2 \tag{3.5}$$

Where ρ_1 is the radius of the cylinder and ρ_2 is the radius of the sphere.

3.6.2.2 Second configuration

The projection of point $\boldsymbol{p_0}$ on the primitive segment is outside the line segment that defines the cylindrical-primitive and is closer to point $\boldsymbol{p_1}$. This case is verified by the following condition:

$$(\boldsymbol{p_0} - \boldsymbol{p_1})^{\mathrm{T}}(\boldsymbol{p_2} - \boldsymbol{p_1}) < 0 \tag{3.6}$$

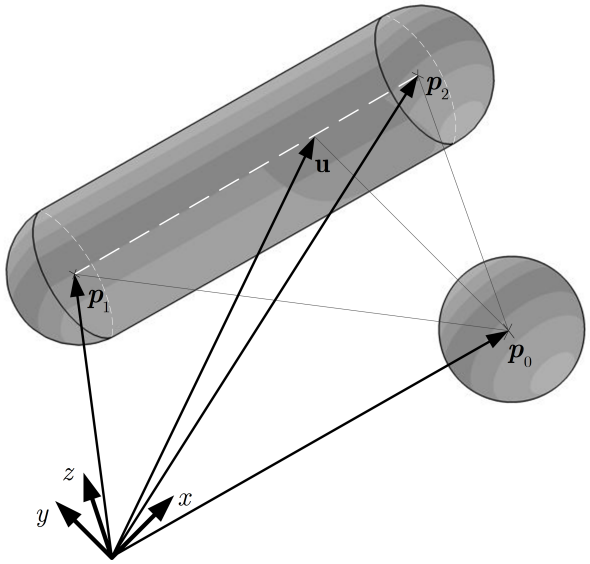

Figure 3.7: Minimum distance between line segment (associated to a cylinder) and a point (center of a sphere)

And the minimum distance is given by:

$$d_{min} = |\boldsymbol{p_1} - \boldsymbol{p_0}| - \rho_1 - \rho_2 \tag{3.7}$$

3.6.2.3 Third configuration

The projection of point $\boldsymbol{p}_0$ on the primitive segment is outside the line segment that defines the primitive and is closer to point $\boldsymbol{p_2}$. This case is verified by the following condition:

$$(\boldsymbol{p_0} - \boldsymbol{p_2})^{\mathrm{T}}(\boldsymbol{p_1} - \boldsymbol{p_2}) < 0 \tag{3.8}$$

And the minimum distance is given by:

$$d_{min} = |\boldsymbol{p_2} - \boldsymbol{p_0}| - \rho_1 - \rho_2 \tag{3.9}$$

3.6.3 Minimum distance between two cylinders

Calculating the minimum distance between two cylinders in an efficient-manner is quite important for our implementation of collision avoidance method. Several researchers have proposed solutions for this purpose. Chapter 3 of [67] describes a method for minimum distance computation between two cylinders with spherical ends. Also in [68] the author presented an algebraic method for this matter. Another method for computing the minimum distance between cylinders with flat ends was proposed in [69] . Nevertheless, the aforementioned methods are lengthy, because they consider the different configurations in which two cylinders might collide with each other. In

[69] seventeen different configurations are considered and in [68] nine different configurations are considered. For the sake of simplicity, in this section we propose a novel solution for calculating the minimum distance between cylinders. This method is based on optimization techniques, it is easy to implement and computationally efficient. To fully describe the proposed method, first we describe the methodology for calculating the common normal on two line segments using least squares technique, followed by reformulation of the minimum distance calculation into a bounded-variable optimization problem. Finally, we propose a novel solution for the minimum distance calculation problem.

3.6.3.1 Common normal on two line segments

In Figure 3.8 it is shown two line segments representing the axis of two primitive cylinders and their associated common normal. Each segment can be defined by two vectors in base frame, one at the beginning of the segment and the other at the end of that segment. Those vectors are denoted by $\boldsymbol{u}_1$, $\boldsymbol{p}_1$ and $\boldsymbol{u}_2$, $\boldsymbol{p}_2$ as shown in Figure 3.8.

It is known that the minimum distance between two lines, $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$, is measured by the length of the common normal on the two segments. It is our objective to find this segment that represents the common normal. For this purpose we describe a simple method that is based on optimization technique and is suitable for computer implementation.

For the calculation of the minimum distance between two line segments, two main cases can be distinguished. The first is when the two lines are not parallel and the common normal is uniquely determined. The other case arose when the two lines are parallel. In such case the common normal is not uniquely determined. We have the freedom to choose one of them, which can be optimized to improve the response of the robot.

Let's designate the position vectors defining the end points of the primitive segment of the first cylinder by $\boldsymbol{p}_1$ and $\boldsymbol{u}_1$, and the position vectors defining the end points of the primitive segment of the second cylinder by $\boldsymbol{p}_2$ and $\boldsymbol{u}_2$. Then, we can define two vectors $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ as:

$$\boldsymbol{s_1} = \boldsymbol{u}_1 - \boldsymbol{p_1} \tag{3.10}$$

And:

$$\boldsymbol{s_2} = \boldsymbol{u_2} - \boldsymbol{p_2} \tag{3.11}$$

While $\boldsymbol{s}_1$ is a vector representing the first primitive segment, $\boldsymbol{s}_2$ is a vector representing the second primitive segment. We can verify whether $\boldsymbol{L_1}$ and $\boldsymbol{L_2}$ are parallel, through the following equality:

$$|\boldsymbol{s_1} \times \boldsymbol{s_2}| = 0 \tag{3.12}$$

For practical implementation, we would like to know whether the two lines are parallel or close to being parallel. Thus, we can use the following inequality:

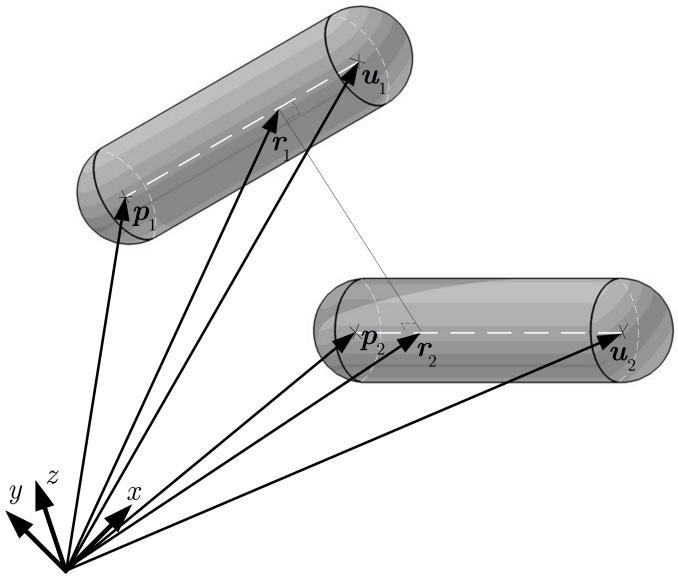

Figure 3.8: Minimum distance between two line segments

$$\frac{abs(\boldsymbol{s_1^T s_2})}{|\boldsymbol{s_1}| \cdot |\boldsymbol{s_2}|} \geq cos(\alpha) \tag{3.13}$$

Where α is the tolerance angle below which the near parallel approximation is applied and *abs* is the absolute value function. If the aforementioned inequality is satisfied the two segments are close to being parallel. Verifying this condition is important because if the two line segments are parallel then the minimum distance between the two segments is constant, and the common normal on both segments is not unique.

To solve the minimum distance problem, when $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ are not parallel, we considered two points of interest on the two primitive segments. Those points are represented relative to base frame by two vectors, $\boldsymbol{r}_2$ and $\boldsymbol{r}_1$, while the parametrized equation of $\boldsymbol{r}_1$ is:

$$\boldsymbol{r_1} = \boldsymbol{p}_1 + \boldsymbol{s_1}\lambda_1 \tag{3.14}$$

And the parametrized equation of $\boldsymbol{r}_2$:

$$\boldsymbol{r_2} = \boldsymbol{p}_2 + \boldsymbol{s_2}\lambda_2 \tag{3.15}$$

Where λ_1 and λ_2 are scalar parameters that have a value in the range from zero to one when the points they represent are confined in between the two ends of the primitive segment. The difference vector $\triangle\boldsymbol{r}$, between $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is:

$$\triangle\boldsymbol{r} = \boldsymbol{r_2} - \boldsymbol{r}_1 \tag{3.16}$$

When the two line segments, $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$, are not parallel, the problem of calculating the minimum distance and their associated points renders to a minimization problem of the norm of vector $\triangle\boldsymbol{r}$:

$$min(|\triangle \boldsymbol{r}|) = min(|\boldsymbol{p_2} + \boldsymbol{s_2}\lambda_2 - (\boldsymbol{p_1} + \boldsymbol{s_1}\lambda_1)|) \tag{3.17}$$

The previous equation is a least squares problem. Its solution is given by the pseudo inverse of the matrix $\mathbf{A}$:

$$\boldsymbol{x_{ls}} = (\mathbf{A^T A})^{-1}\mathbf{A^T}\boldsymbol{y} \tag{3.18}$$

We mention here that $\mathbf{A^T A}$ is invertible since that $\mathbf{A}$ is full rank, which is assured by the assumption that $\boldsymbol{L_1}$ and $\boldsymbol{L_2}$ are not parallel, or in other words, the column vectors of $\mathbf{A}$ are independent. The solution $\boldsymbol{x_{ls}}$ can be interpreted as the coefficients associated with the projection of $\boldsymbol{y}$ on $R(\mathbf{A})$, where $R(\mathbf{A})$ is the range of $\mathbf{A}$. Thus, the minimum distance between the two segments, d_{min}, is the norm of the residual minus the radius of the two cylinders. The residual is the difference between vector $\boldsymbol{y}$ and its projection $\mathbf{A}\boldsymbol{x_{ls}}$. d_{min} is given by:

$$d_{min} = |(\mathbf{1_3} - \mathbf{A}(\mathbf{A^T A})^{-1}\mathbf{A^T})\boldsymbol{y}| - \rho_1 - \rho_2 \tag{3.19}$$

The vectors $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ associated with the minimum distance are calculated from:

$$\boldsymbol{r_1} = \boldsymbol{p_1} + \mathbf{A}\begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}\boldsymbol{x_{ls}} \tag{3.20}$$

$$\boldsymbol{r_2} = \boldsymbol{p_2} + \mathbf{A}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\boldsymbol{x_{ls}} \tag{3.21}$$

Thus, the common normal and its end points are fully specified.

3.6.3.2 Quadratic optimization

In this section we present a method for calculating the minimum distance between line segments, taking into consideration that the optimization variables of the previous least squares problem are bounded. As shown in the previous section, the minimum distance calculation between two line segments reduces to the following optimization problem:

$$min(|\triangle \boldsymbol{r}|) = min(|\mathbf{A}\boldsymbol{x} - \boldsymbol{y}|) \tag{3.22}$$

Where matrix $\mathbf{A}$ is the concatenation of vectors $\boldsymbol{s_1}$ and $\boldsymbol{s_2}$:

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{s_2} & -\boldsymbol{s_1} \end{bmatrix} \tag{3.23}$$

And vector $\boldsymbol{y}$ is:

$$\boldsymbol{y} = \boldsymbol{p_1} - \boldsymbol{p_2} \tag{3.24}$$

We can rewrite the optimization function in the following equivalent quadratic form:

$$min(f) = min((\mathbf{A}\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}}(\mathbf{A}\boldsymbol{x} - \boldsymbol{y})) \tag{3.25}$$

And the problem can be viewed as minimizing the previous function, subject to the following constrains:

$$0 < x_1 < 1 \tag{3.26}$$

And:

$$0 < x_2 < 1 \tag{3.27}$$

Where x_1 and x_2 are the components of the vector $\boldsymbol{x}$. We can reformulate the function f by performing **QR** factorization on matrix **A**. Then, the optimization function can be rewritten:

$$f = (\mathbf{QR}\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}}(\mathbf{QR}\boldsymbol{x} - \boldsymbol{y}) \tag{3.28}$$

Where **Q** is a 3×2 orthogonal matrix and **R** is a 2×2 upper triangular matrix. The previous function can be manipulated by taking advantage of the fact that $\mathbf{Q^T Q = 1_2}$. Thus, after manipulation and fixing we find that minimizing (3.25) is equivalent to minimizing the function:

$$f = (\mathbf{R}\boldsymbol{x} - \mathbf{Q^T}\boldsymbol{y})^{\mathrm{T}}(\mathbf{R}\boldsymbol{x} - \mathbf{Q^T}\boldsymbol{y}) \tag{3.29}$$

Or we can rewrite:

$$min(f) = min(\boldsymbol{u}^{\mathrm{T}}\boldsymbol{u}) \tag{3.30}$$

Where $\boldsymbol{u}$ is given by:

$$\boldsymbol{u} = (\mathbf{R}\boldsymbol{x} - \mathbf{Q^T}\boldsymbol{y}) \tag{3.31}$$

Thus, the region of feasible solutions can be transformed using the transformation function (3.31). There are several points that we can notice from (3.31). First, the transform function is an affine function and the transformation matrix **R** is an upper triangular matrix. The rectangular region of feasible solutions becomes a parallelogram region after the transformation. Second, due to the fact that the transform is an affine and the original feasible solution region (rectangular region) is a convex set, then the resulting region after the transform is also a convex set [70]. Figure 3.9 and Figure 3.10 show the rectangular regions before the transform and the parallelogram regions after the transform, for two different cases.

After applying the transform, the bounded variable optimization problem becomes equivalent to searching for that point within the parallelogram region which is closest to the origin. According to the relative position of the parallelogram and the origin, we can distinguish two different situations. In the first the origin is inside the parallelogram, the closest point of the parallelogram region to the origin is the origin itself, and the minimum value of the modified objective function (3.29) is zero. In this situation the problem's solution is:

$$\boldsymbol{u} = \mathbf{0} \Longrightarrow \mathbf{R}\boldsymbol{x} - \mathbf{Q^T}\boldsymbol{y} = \mathbf{0} \tag{3.32}$$

Equivalently:

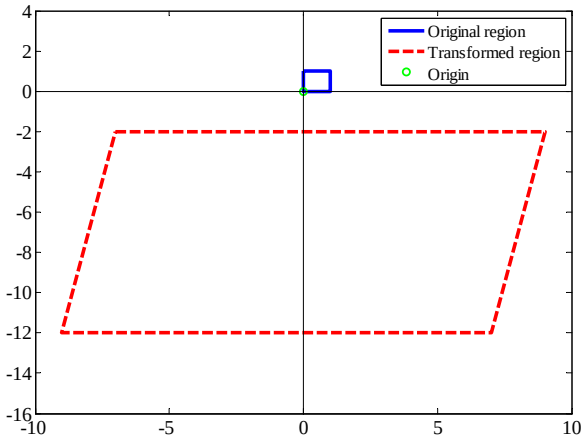

Figure 3.9: Region of feasible solutions for the quadratic optimization problem. The origin is outside the transformed region.

$$\boldsymbol{x} = \mathbf{R}^{-1}\mathbf{Q}^{\mathbf{T}}\boldsymbol{y} \tag{3.33}$$

We notice that the previous solution is equal to the least squares solution (3.18) of the unbounded least-squares problem, $\boldsymbol{x}_{min} = \boldsymbol{x}_{ls}$, which is easily verifiable by blogging $\mathbf{A} = \mathbf{QR}$ into least squares solution (3.18). The second situation occurs when the origin is outside the parallelogram. In this case the solution is the point of the parallelogram closest to the origin. If we denoted the position vector of the point of the parallelogram closest to the origin by $\boldsymbol{u}_{\boldsymbol{min}}$, which also represents the solution-vector of the modified optimization problem (3.30), then the solution-vector of the original problem (3.25) can be calculated from:

$$\boldsymbol{x}_{\boldsymbol{min}} = \mathbf{R}^{-1}(\boldsymbol{u}_{\boldsymbol{min}} + \mathbf{Q}^{\mathbf{T}}\boldsymbol{y}) \tag{3.34}$$

Once the parameters vector $\boldsymbol{x}_{\boldsymbol{min}}$ is calculated, the minimum distance can be computed:

$$d_{min} = |\mathbf{A}\boldsymbol{x}_{\boldsymbol{min}} - \boldsymbol{y}| - \rho_1 - \rho_2 \tag{3.35}$$

The point of the robot's primitive closest to the obstacle can be calculated from (3.36) and the point of the obstacle's primitive closest to the robot can be calculated from (3.37).

$$\boldsymbol{r}_{\mathbf{1}} = \boldsymbol{p}_{\mathbf{1}} + \mathbf{A}\begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}\boldsymbol{x}_{\boldsymbol{min}} \tag{3.36}$$

$$\boldsymbol{r}_{\mathbf{2}} = \boldsymbol{p}_{\mathbf{2}} + \mathbf{A}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\boldsymbol{x}_{\boldsymbol{min}} \tag{3.37}$$

The minimum distance between the two cylinders, d_{min}, can be computed without a need of inverting $\mathbf{R}$. This can speed up calculations in applications where collision

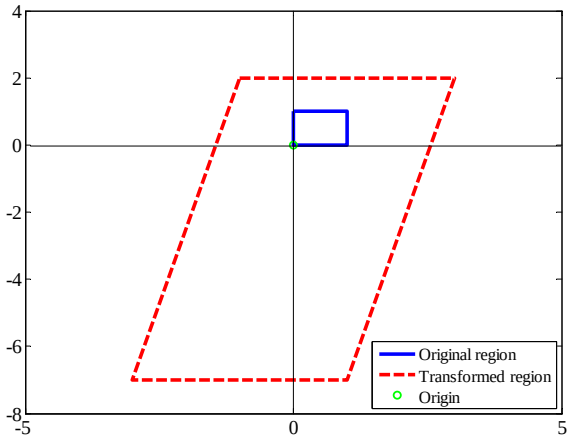

Figure 3.10: Region of feasible solutions for the quadratic optimization problem. The origin is inside the transformed region.

detection is the main objective. This is achieved by substituting the solution of the bounded variable optimization problem (3.34) into equation (3.35):

$$d_{min} = |\mathbf{QRR^{-1}}(\boldsymbol{u_{min}} + \mathbf{Q^T}\boldsymbol{y}) - \boldsymbol{y}| - \rho_1 - \rho_2 \tag{3.38}$$

By eliminating the term $\mathbf{RR^{-1}}$:

$$d_{min} = |\mathbf{Q}(\boldsymbol{u_{min}} + \mathbf{Q^T}\boldsymbol{y}) - \boldsymbol{y}| - \rho_1 - \rho_2 \tag{3.39}$$

Thus, from the previous expression, the minimum distance can be computed directly after retrieving the point $\boldsymbol{u_{min}}$ closest to the origin.

Algorithm 3.1 Bounded variable least-sequares optimization for minimum distance calculation

Input : $\boldsymbol{\mathcal{R}}$ region of feasible solutions
$(\mathbf{Q}, \mathbf{R})$ facorization matrices of $\mathbf{A}$
Output : $\boldsymbol{x_{min}}$ vector of coefficients $[\lambda_1, \lambda_2]^{\mathrm{T}}$ associated with minimum distance

01 : Transform $\boldsymbol{\mathcal{R}}$ using the function $\boldsymbol{f(x)} = (\mathbf{R}\boldsymbol{x} - \mathbf{Q^T}\boldsymbol{y})$
02 : **If** Origin is inside $\boldsymbol{\mathcal{R}}$ **then**
03 : $\boldsymbol{u_{min}} \leftarrow [0, 0]^{\mathrm{T}}$
04 : **else**
05 : **for** each boundary segment of $\boldsymbol{\mathcal{R}}$ **do**
06 : $\boldsymbol{c} \leftarrow$ point of segment closest to origin
07 : **If** first iteration **then**
08 : $\boldsymbol{u_{min}} \leftarrow \boldsymbol{c}$
09 : **else**
10 :: **If** norm($\boldsymbol{u_{min}}$) > norm($\boldsymbol{c}$) **then**
11 : $\boldsymbol{u_{min}} \leftarrow \boldsymbol{c}$
12 : **end if**
13 : **end if**
14 : **end for**
15 : **end if**
16 : $\boldsymbol{x_{min}} \leftarrow \mathbf{R^{-1}}(\boldsymbol{u_{min}} + \mathbf{Q^T}\boldsymbol{y})$

3.6.3.3 Conclusion:

In this section we proposed a novel method to solve the minimum distance between two cylinders. We reformulated the problem as a bounded variable optimization problem. For solving the optimization problem we performed an affine-transformation on the region of feasible solutions. This transformation was deduced from **QR** decomposition of matrix **A**. After the transform, the region of feasible solutions becomes a parallelogram, and remains convex. We showed that the solution to the optimization problem corresponds to the point of the parallelogram region closest to the origin. Figure 3.11 and Figure 3.12 give a visual illustration of the idea. Figure 3.11 shows the optimization problem, where ellipses represent level sets of the cost function and the rectangle represents the region of feasible solutions. Figure 3.12 shows the equivalent optimization problem after applying the proposed transform. The contour ellipses and the level sets of the original cost function are transformed into concentrated circles. The rectangular region of feasible solutions was transformed into a parallelogram. In the example illustrated in Figure 3.11 and Figure 3.12, the origin is inside the parallelogram region resulting after the transformation, so we deduced that the solution of the bounded-variable least squares problem is coincident with the origin, and as we showed previously, the solution attained for this case shall be equal to the solution of the unbounded least squares problem. Otherwise, the solution would be the point of the parallelogram boundary closest to the origin.

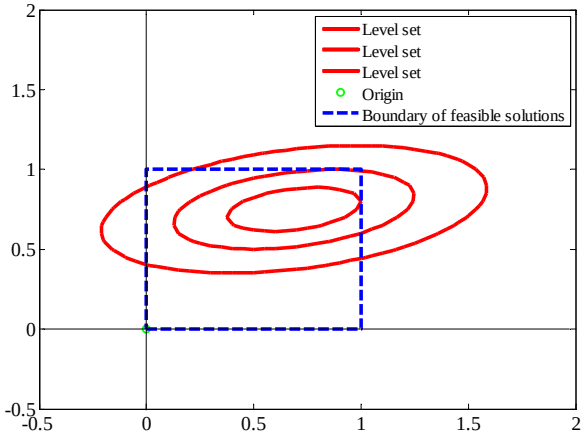

Figure 3.11: The original optimization problem defined by level sets of objective function and region of feasible solutions, before applying the transform.

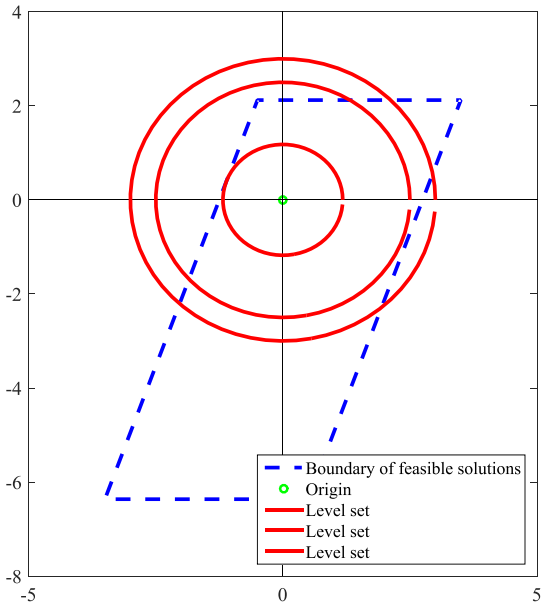

Figure 3.12: Equivalent optimization problem defined by level sets of equivalent objective function and region of feasible solutions, after the transform.

3.7 Mathematical model of the robot

To control the robot, a mathematical model of its kinematics and/or dynamics shall be established. Robot's configuration is described by its joints angular position vector

$\boldsymbol{q}$. The mapping from the robot's configuration $\boldsymbol{q}$ to the end-effectors position $\boldsymbol{x}$ is established by the direct kinematics:

$$\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{q}) \tag{3.40}$$

Where $\boldsymbol{f}$ is the manipulator's forward kinematics, a highly non-linear vector function, that gives the position and orientation of the end-effector as a function of joints angles. This function can be calculated using the homogeneous transformation matrices, following Denavit-Hartenberg convention [13], [4]. Differentiating (3.40) gives the equation of differential kinematics:

$$\dot{\boldsymbol{x}} = \mathbf{J}\dot{\boldsymbol{q}} \tag{3.41}$$

Where $\dot{\boldsymbol{q}}$ is the joints velocities vector, $\dot{\boldsymbol{x}}$ is the end-effector's velocity and $\mathbf{J}$ is the manipulator's Jacobean:

$$J_{ij} = \frac{\delta f_i}{\delta q_j} \tag{3.42}$$

$\mathbf{J}$ is a function of links geometry and the robots' configuration. Regarding dynamics modelling, the subject was fully treated in the dynamics chapter of this study.

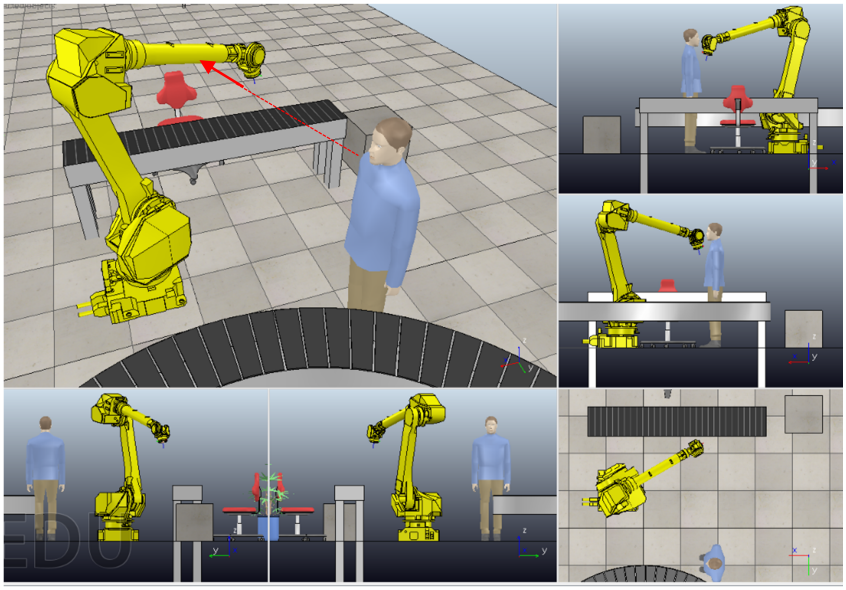

Figure 3.13: Repulsion vector aligned with minimum distance segment

3.8 Vectors influencing robot's motion

To achieve robot's motion that satisfies the collision avoidance objective while adhering as possible to the original trajectory, the concept of hypothetical repulsion and attraction vectors will be introduced. Depending on the type of control used, those vectors represent velocities, or forces, that attract the robot towards its goal while repelling it away from obstacles. Those vectors, their physical meaning and their mathematical representation will be analysed.

3.8.1 Repulsion vector

After computing the minimum distance between the obstacle and the robot, and after calculating the point of the robot primitive associated with the minimum distance, the repulsion vector has to be calculated. The repulsion vector is defined by amplitude and direction, Figure 3.13. The direction of the repulsion vector $\boldsymbol{s}$ will be aligned with the line segment associated with minimum distance. If we designate the position vector of that point of the obstacle closest to the robot by $\boldsymbol{p_o}$ and the point of the robot closest to the obstacle by $\boldsymbol{p_r}$, then, the vector $\boldsymbol{s}$ is given by:

$$\boldsymbol{s} = \frac{\boldsymbol{p_r} - \boldsymbol{p_o}}{|\boldsymbol{p_r} - \boldsymbol{p_o}|} \tag{3.43}$$

The magnitude of the repulsion vector could be calculated based on one of several measures that quantify the risk of collision:

- The minimum distance between the obstacle and the robot, [39].
- Time to collision, Chapter 5 of [71]. This parameter was calculated from the

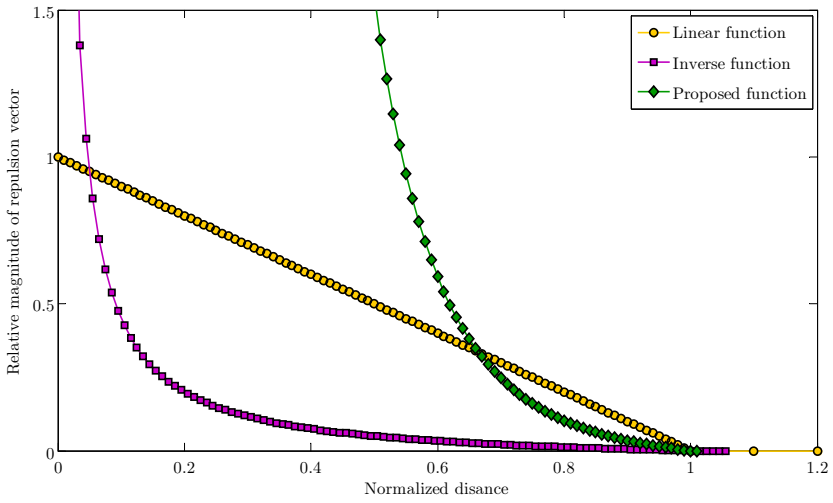

Figure 3.14: Magnitude of normalized repulsion vector as function of normalized distance using different functions.

minimum distance and an estimation of the relative velocity between the robot and the obstacle.

- An estimation of the acceleration required to achieve collision avoidance, [72].

In this study we choose the minimum distance as the measure of collision imminence. For successful collision avoidance strategy, the repulsion vector is calculated such that its magnitude increases when the minimum distance decreases. Based on this criteria, three functions for calculating the magnitude of the repulsion vector had been proposed:

1- The inverse of the minimum distance is usually used for calculating the magnitude of the repulsion vector. This function was used in [39] and is given by:

$$f_{rep} = \begin{cases} k\left(\frac{1}{d_{min}} - \frac{1}{d_0}\right), & if\ d_{min} < d_0 \\ 0 & if\ d_{min} \geq d_0 \end{cases} \tag{3.44}$$

Where d_0 is the safety-zone parameter or the distance at which the repulsion vector is activated. We notice that the magnitude of the repulsion vector increases to infinity when d_{min} goes to zero. To avoid saturation on motors the resulting joints action values, torques in case of revolute joints, need to be clamped to the maximum permissible value.

2- Another function that is also used to calculate the magnitude of the repulsion vector is given by a linear function of the minimum distance:

$$f_{rep} = \begin{cases} k\left(1 - \frac{d_{min}}{d_0}\right), & if\ d_{min} < d_0 \\ 0 & if\ d_{min} \geq d_0 \end{cases} \tag{3.45}$$

3- In Chapter 6 of [73] the author selected a cosine shaped function to calculate the repulsive vectors.

It is worth noting that during the course of this study other functions, in addition to the aforementioned, were utilized, and a good response have been achieved using the following function:

$$f_{rep} = \begin{cases} k\left(\left(\frac{d_0}{d_{min}}\right)^5 - 1\right), & if\ d_{min} < d_0 \\ 0 & if\ d_{min} \geq d_0 \end{cases} \tag{3.46}$$

For the tests performed in this study, the above function was utilized. Figure 3.14 shows different curves representing the aforementioned force functions.

3.8.2 Attraction vector

An attraction vector attached to the end-effector allows the robot to follow the goal. This vector has the function of guiding the end-effector towards the goal point. During this study, a proportional - integral (PI) controller was utilized for computing the end-effector's attraction vector. The error vector is defined as the difference between the end-effector's position and the attraction pole position:

$$\boldsymbol{e} = \boldsymbol{p_e} - \boldsymbol{p_{pole}} \tag{3.47}$$

Where $\boldsymbol{e}$ is the error vector, represented by a dotted arrow in Figure 3.15, $\boldsymbol{p}_e$ is the end-effector's position, notated (TCP) for tool center point, and $\boldsymbol{p_{pole}}$ is the position vector of the attraction pole on the path. The attraction vector is calculated from the PI equation:

$$\boldsymbol{v_{e.att}} = -\mathbf{K_p}\boldsymbol{e} - \mathbf{K_i}\int \boldsymbol{e}dt \tag{3.48}$$

Where $\boldsymbol{v_{e.att}}$ is the attraction vector acting on the TCP and it has a physical meaning of velocity. The subscript *att* denotes that this is an attraction vector, e is used to denote that the vector is acting on the end-effector, $\mathbf{K_p}$ is the proportionality coefficient matrix and $\mathbf{K_i}$ is the integral term coefficient matrix.

3.9 Collision avoidance controllers

Control of robots can be performed using one of two strategies, kinematics based control and the force based control [74]. In the context of this study, and following those two strategies, two different types of controllers with collision avoidance capabilities were implemented, Figure 3.16. We will designate the controller based on kinematics by (K) controller, and the controller based on force control by (F) controller.

3.9.1 Kinematics control

In this case the control acts at the kinematics level, as such the attraction vector is treated as a velocity that causes the motion of the end-effector towards the goal, while

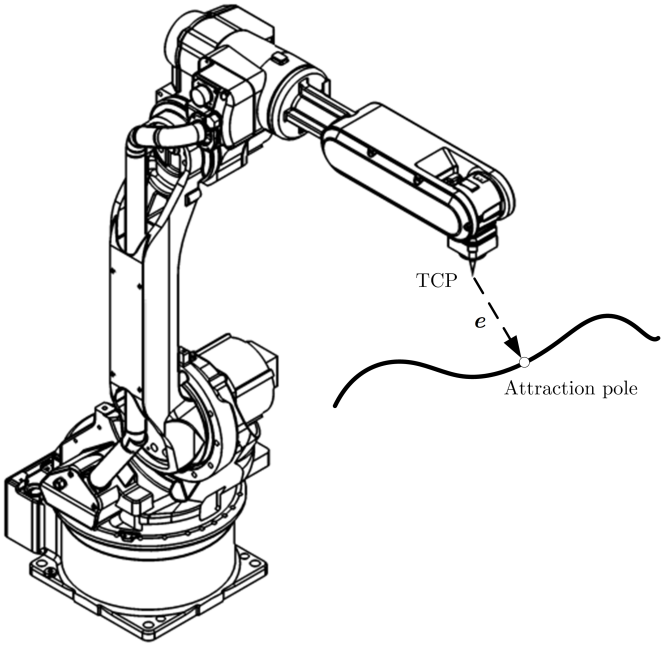

Figure 3.15: The path curve, the attraction pole, and the error vector

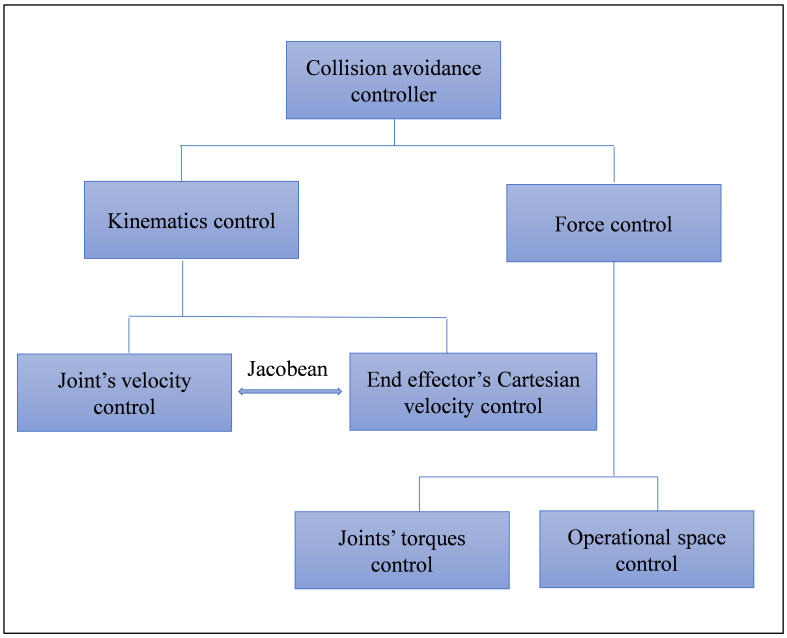

Figure 3.16: Outline of collision avoidance strategies

the repulsion vector is treated as repulsion velocity. This repulsion velocity acts on the point of the robot associated with the minimum distance and repel it away from collision.

For controlling the robot in the joint space, the corresponding joints velocities have to be calculated. This conversion from Cartesian velocities to joint space velocities is accomplished by utilizing the Jacobean inverse, or to a better extent, the generalized inverses of the Jacobeans associated with the control point and the end-effector. Note that the *control point* (CP) is the point of the robot closest to the obstacle.

On the other hand, when the robot is being controlled in operational space, as in this study, the repulsion velocities shall be propagated from the control point associated with the minimum distance up to the robot's end-effector. In this case, the operational space control command utilized is the velocity of the end-effector described in operational space. This command is calculated as the vector sum of the attraction velocity, acting at the end-effector, and the repulsion velocity, propagated from CP up to the end-effector. In this study the attraction velocity is calculated by the PI relation given in (3.48). The propagation of the repulsion velocity, from the control point associated with the minimum distance up to the end-effector, is performed by utilizing the Jacobean and inverse kinematics. There are several techniques proposed in the literature for performing inverse kinematics calculation. Comprehensive review about the subject is given in [75] and a performance-comparison of the most important methods for computing inverse kinematics is presented in [76]. In this study two techniques were utilized, one is the Jacobean transpose and the other is the damped least squares (DLS). Despite the fact that the DLS is more costly in terms of computation, the method offers better stability than the Jacobean transpose method, which is lighter to compute.

The DLS solution for propagating the velocity from the control point up to the end-effector can be deduced as follows. First the angular velocities $\dot{\boldsymbol{q}}_{\boldsymbol{rep}}$ associated with the repulsion velocity vector are calculated:

$$\dot{\boldsymbol{q}}_{\boldsymbol{rep}} = \mathbf{J}_{\mathbf{cp}}^{\mathbf{T}}(\mathbf{J}_{\mathbf{cp}}\mathbf{J}_{\mathbf{cp}}^{\mathbf{T}} + \lambda^2\mathbf{1}_{\boldsymbol{m}})^{-1}\boldsymbol{v}_{\boldsymbol{cp.rep}} \tag{3.49}$$

Where $\dot{\boldsymbol{q}}_{\boldsymbol{rep}}$ is the repulsion angular velocities vector, $\mathbf{J}_{\mathbf{cp}}$ is the Jacobean associated with the control point on the robot, λ is a damping constant, $\mathbf{1}_{\boldsymbol{m}}$ is the identity matrix and $\boldsymbol{v}_{\boldsymbol{cp.rep}}$ is the repulsion velocity at the control point. Then, the propagated velocity from the CP up to the end-effector is given by:

$$\boldsymbol{v}_{\boldsymbol{e.rep}} = \mathbf{J}_{\mathbf{e}}\dot{\boldsymbol{q}}_{\boldsymbol{rep}} \tag{3.50}$$

Where $\boldsymbol{v}_{\boldsymbol{e.rep}}$ is the repulsion velocity transferred to the end-effector and $\mathbf{J}_{\mathbf{e}}$ is the Jacobean associated with the end-effector. Another way to calculate the propagation of the repulsion velocity to the end-effector is by using the Jacobean transpose:

$$\boldsymbol{v}_{\boldsymbol{e.rep}} = \alpha\mathbf{J}_{\mathbf{e}}\mathbf{J}_{\mathbf{cp}}^{\mathbf{T}}\boldsymbol{v}_{\boldsymbol{cp.rep}} \tag{3.51}$$

Where α is a scalar, which according to [75] is defined by:

$$\alpha = \frac{\boldsymbol{v}_{\boldsymbol{cp,rep}}^{\mathbf{T}} \mathbf{J}_{\mathbf{cp}} \mathbf{J}_{\mathbf{cp}}^{\mathbf{T}} \boldsymbol{v}_{\boldsymbol{cp.rep}}}{\boldsymbol{v}_{\boldsymbol{cp,rep}}^{\mathbf{T}} \mathbf{J}_{\mathbf{cp}} \mathbf{J}_{\mathbf{cp}}^{\mathbf{T}} \mathbf{J}_{\mathbf{cp}} \mathbf{J}_{\mathbf{cp}}^{\mathbf{T}} \boldsymbol{v}_{\boldsymbol{cp,rep}}} \tag{3.52}$$

From [75], for a faster calculation of α, an intermediary vector ε can be defined:

$$\boldsymbol{\varepsilon} = \mathbf{J}_{\mathbf{cp}} \mathbf{J}_{\mathbf{cp}}^{\mathbf{T}} \boldsymbol{v}_{\boldsymbol{cp.rep}} \tag{3.53}$$

Then, the expression of α reduces to:

$$\alpha = \frac{\boldsymbol{\varepsilon}^{\mathbf{T}} \boldsymbol{v}_{\boldsymbol{cp.rep}}}{\boldsymbol{\varepsilon}^{\mathbf{T}} \boldsymbol{\varepsilon}} \tag{3.54}$$

Thus, the total operational space command sent to the robot is the summation of two vectors:

$$\boldsymbol{v_e} = \boldsymbol{v_{e.att}} + \boldsymbol{v_{e.rep}} \tag{3.55}$$

For the purpose of this study and to achieve better numerical stability, we follow the recommendation of [75], where the alternate Jacobean was utilized. The alternate Jacobean is the Jacobean associated with the point of the goal position, rather than the robot's end-effector. To give an intuition about the physical meaning of this formulation, the reader can think of it as trying to move the goal position toward the end-effector, rather than the end-effector toward the goal position. A complete description along with mathematical formulation of the alternate Jacobean is in [75].

3.9.2 Force control

In this case the dynamics of the robot is taken into consideration when generating collision avoidance motions. The repulsion vectors acting on the points associated with the minimum distance are treated as forces. These forces shall be transformed into control commands. The method of transformation depends on the way in which the control is performed. We can distinguish between two types of control, joint space control and operational space control.

3.9.2.1 Joint space control

This method is used in [41], where control commands are calculated in joint space, and each command is composed of the joint generalized forces. Joint generalized forces is a synonym for joints torques in case of robot with all revolute joints. In this case, the control command is calculated as the summation of two different torques:

1- Torque command that pulls the robot toward the goal configuration. We can use off-the-shelf proportional-derivative (PD) controller to achieve this result, as used in [41]:

$$\boldsymbol{\tau_{att}} = -\mathbf{K}_{\boldsymbol{p}}(\boldsymbol{q} - \boldsymbol{q_d}) - \mathbf{K}_{\boldsymbol{d}}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_{\boldsymbol{d}}) \tag{3.56}$$

Where $\boldsymbol{\tau_{att}}$ is the joints torques vector that drives the robot towards the goal configuration, $\boldsymbol{q_d}$ is the goal's configuration, $\dot{\boldsymbol{q}}_{\boldsymbol{d}}$ is the vector of the desired joints' velocities

at goal configuration, and coefficients $\mathbf{K_p}$ and $\mathbf{K_d}$ can be adjusted as the robot gets closer to the goal. This equation shed a light on the drawback of this method, due to the fact that robot's tasks are usually described in the Cartesian space. Thus, $\boldsymbol{q_d}$ and $\dot{\boldsymbol{q}}_{\boldsymbol{d}}$ need to be calculated by transferring the trajectory from the Cartesian into the joint space domain. This conversion is computationally expensive.

2- Torque command that pushes the robot away from the obstacle:

$$\boldsymbol{\tau_{rep}} = \mathbf{J_{cp}^T} \boldsymbol{f_{rep}} \tag{3.57}$$

Where $\mathbf{J_{cp}^T}$ is the transpose of the Jacobean associated with the point of the robot closest to the obstacle and $\boldsymbol{f_{rep}}$ is the repulsion force calculated at that point. Then, the total control torque command is:

$$\boldsymbol{\tau_c} = \boldsymbol{\tau_{att}} + \boldsymbol{\tau_{rep}} \tag{3.58}$$

To avoid saturations on the motors, the torques are clamped by the maximum permissible torque of the motors:

$$\bar{\boldsymbol{\tau}}_{\boldsymbol{c}} = \mathbf{C}\boldsymbol{\tau_c} + (\mathbf{1_n} - \mathbf{C})\boldsymbol{\tau_{max}} \tag{3.59}$$

Where $\bar{\boldsymbol{\tau}}_{\boldsymbol{c}}$ is the truncated control-torque command, $\mathbf{1_n}$ is the identity matrix, n is the DOF of the robot, and $\mathbf{C}$ is an $n \times n$ diagonal matrix:

$$c_{ij} = \begin{cases} 1 & i = j,\ \tau_{c.i} < \tau_{max,i} \\ 0 & i = j,\ \tau_{c.i} \geq \tau_{max,i} \\ 0 & i \neq j \end{cases} \tag{3.60}$$

Where $\boldsymbol{\tau_c}$ is the i^{th} element of the vector $\boldsymbol{\tau_c}$ and $\boldsymbol{\tau_{max}}$ is the maximum permissible torque that can be applied on link i. If we have a torque controlled robot, then motor-torques commands are applied directly to robot's controllers after performing the proper transformation on torques from joint space to actuator space. In other situations joints angles have to be calculated. This is done by integrating robot direct-dynamics equation.

3.9.2.2 Operational space force control

In this type of control the robot is controlled in Cartesian space by specifying motion parameters of a unit-mass of the decoupled end-effector [7]. In this case, the dynamics equations are referenced to the end-effector.

We recall that our control is acting under the influence of an attraction force that acts on the end-effector, attracting it towards the goal, and a repulsion force that acts on the point of the robot associated with the minimum distance to the obstacle. Thus, the operational space representation of our problem would be to reference both of these vectors, attraction and repulsion, to the end-effector.

The first part of the problem, referencing the attraction vector to the end-effector, is done through the application of an attraction force on a single unit-mass that represents

the decoupled end-effector. This method is described in [39], where the author used proportional derivative (PD) control to calculate the attraction force:

$$\boldsymbol{f_{e.att}} = -\mathbf{K_p}(\boldsymbol{x} - \boldsymbol{x_d}) - \mathbf{K_v}(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_{\boldsymbol{d}}) \tag{3.61}$$

We note that the force acting on the decoupled unit-mass is equal to its acceleration, so $\boldsymbol{f_{e.att}}$ has the interpretation of a force and acceleration.

The repulsion force that acts on the point of the manipulator closest to the obstacle shall be transferred to the end-effector. We saw previously that in the case of the K controller the transformation of the repulsion velocity was done by utilizing the Jacobean. However, in the case of the F controller the transformation law shall take into consideration the dynamics of the manipulator. The transformation law can be deduced from the equation of the inverse dynamics of the manipulator's motion:

$$\boldsymbol{\tau} = \mathbf{A}\ddot{\boldsymbol{q}} + \mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g} \tag{3.62}$$

To describe the equation of motion of the end-effector in the operational space, it is necessary to reformulate the equation of the inverse-dynamics to include the acceleration of the end-effector $\ddot{\boldsymbol{x}}$, instead of joints accelerations $\ddot{\boldsymbol{q}}$. To do this we need an equation that relates $\ddot{\boldsymbol{x}}$ and $\ddot{\boldsymbol{q}}$ together. This equation is deduced through the differentiation of the end-effector's Cartesian velocity expression. The end-effector's Cartesian velocity is given by:

$$\dot{\boldsymbol{x}} = \mathbf{J_e}\dot{\boldsymbol{q}} \tag{3.63}$$

By differentiation with respect to time:

$$\ddot{\boldsymbol{x}} = \mathbf{J_e}\ddot{\boldsymbol{q}} + \frac{d\mathbf{J_e}}{dt}\dot{\boldsymbol{q}} \tag{3.64}$$

Or:

$$\ddot{\boldsymbol{q}} = \mathbf{J_e^{-1}}\left(\ddot{\boldsymbol{x}} - \frac{d\mathbf{J_e}}{dt}\dot{\boldsymbol{q}}\right) \tag{3.65}$$

Then, the equation of the inverse-dynamics (3.62) can be reformulated in terms of the end-effector's acceleration as:

$$\boldsymbol{\tau} = \mathbf{A}\mathbf{J_e^{-1}}\left(\ddot{\boldsymbol{x}} - \frac{d\mathbf{J_e}}{dt}\dot{\boldsymbol{q}}\right) + \mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g} \tag{3.66}$$

To deduce a relationship between the motion of the unit-mass of the decoupled end-effector and the virtual forces acting on the robot we apply the following principal:

$$[Total\ torque] = [Follow\ target\ torque] + [Collision\ avoidance\ torque]$$

Which can be translated into the following equation:

$$\mathbf{A}\mathbf{J_e^{-1}}\left(\ddot{\boldsymbol{x}} - \frac{d\mathbf{J_e}}{dt}\dot{\boldsymbol{q}}\right) + \mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g} =$$

$$\left\{\mathbf{A}\mathbf{J}_e^{-1}\left(\boldsymbol{f_{e.att}} - \frac{d\mathbf{J}_e}{dt}\dot{\boldsymbol{q}}\right) + \mathbf{B}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}\right\} + \left\{\mathbf{J}_{\mathbf{cp}}^{\mathbf{T}}\boldsymbol{f_{rep}}\right\} \tag{3.67}$$

It is important to mention that the unknown of the aforementioned equation is $\ddot{\boldsymbol{x}}$, or the resulting motion of the unit-mass of the decoupled end-effector. By simplifying the aforementioned equation we find that the total acceleration of the unit-mass of the decoupled end-effector $\ddot{\boldsymbol{x}}$ is:

$$\ddot{\boldsymbol{x}} = \boldsymbol{f_{e.att}} + \mathbf{J}_e\mathbf{A}^{-1}\mathbf{J}_{\mathbf{cp}}^{\mathbf{T}}\boldsymbol{f_{rep}} \tag{3.68}$$

Where $\ddot{\boldsymbol{x}}$ satisfies the requirement of a motion that is as close as possible to the predefined trajectory, while avoiding collision with the obstacle.

3.9.2.3 Linearising force control equation

To deduce an expression of a velocity-level control that takes into consideration robot's dynamics, and starting from (3.68), we notice that the equation can be linearized near the current position of the end-effector. By using forward Euler approximation, we can re-write the acceleration of the end-effector:

$$\ddot{\boldsymbol{x}} = \frac{\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_{\mathbf{0}}}{\triangle t} \tag{3.69}$$

Where $\dot{\boldsymbol{x}}$ is the end-effector's velocity that achieves collision avoidance while trying to adhere as possible to the predefined trajectory, $\dot{\boldsymbol{x}}_{\mathbf{0}}$ represents the current velocity of the end-effector, and $\triangle t$ is the simulation update interval. By applying the same notion, the force of attraction $\boldsymbol{f_{e.att}}$ can be approximated by:

$$\boldsymbol{f_{e.att}} = \frac{\dot{\boldsymbol{x}}_{\boldsymbol{e.att}} - \dot{\boldsymbol{x}}_{\mathbf{0}}}{\triangle t} \tag{3.70}$$

Here $\boldsymbol{f_{e.att}}$ is interpreted as the acceleration of the decoupled unit mass of the end-effector. By substituting (3.69) and (3.70) into (3.68), and fixing, we see that the linearized equation of motion becomes:

$$\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}}_{\boldsymbol{e.att}} + \mathbf{J}_e\mathbf{A}^{-1}\mathbf{J}_{\mathbf{cp}}^{\mathbf{T}}\boldsymbol{f_{rep}}\triangle t \tag{3.71}$$

For calculating $\dot{\boldsymbol{x}}_{\boldsymbol{e.att}}$ a PI controller (3.48), was utilized. The aforementioned equation has a higher numerical stability than inverting the Jacobean. Since that $\mathbf{A}$ is symmetric positive definite it is always invertible [77]. In addition, the resulting motion have the added advantage of taking into consideration robot's dynamics.

3.10 Other considerations

When calculating control commands it is not satisfactory to avoid collisions with obstacles only. There are other aspects that shall be taken into consideration, such as self-collisions due to robot's links colliding with each other, joints limits, and work-space limits which all shall be avoided. In the following subsections we describe how to incorporate the aforementioned considerations into the collision avoidance method.

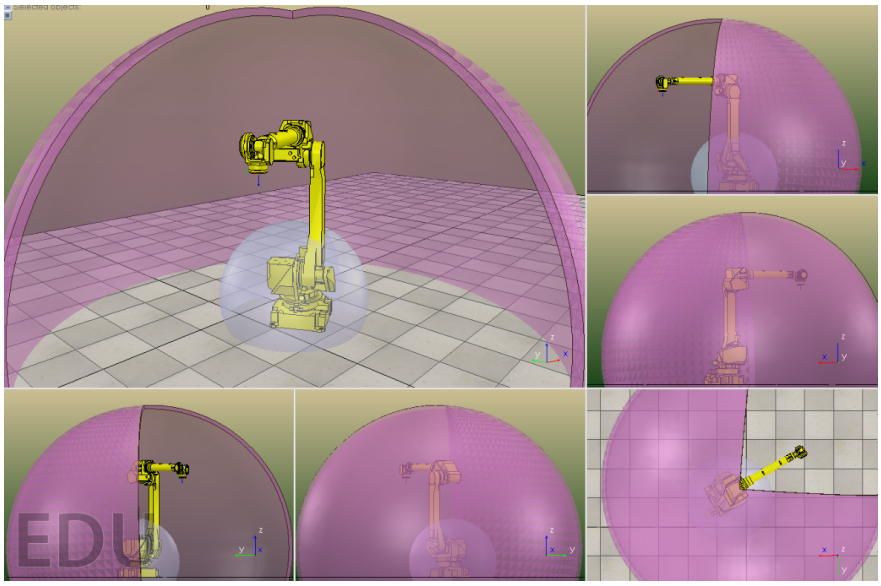

Figure 3.17: Work space limits approximated as spheres in operational space

3.10.1 Workspace limit

Workspace limits can be addressed directly in operational space. This is done by adding virtual repulsion surfaces to the environment model. These surfaces act on the robot with repulsion forces when it comes closer to work space limits. In this study the workspace is approximated by two half spheres and a plane, Figure 3.17. Half sphere for modelling the interior limit of the workspace, half sphere for the exterior limit of the workspace and a flat plane for the floor. These limiting surfaces are considered as sources of repulsion potentials. These potentials act on robot's links when they reach some predefined safety margin from workspace limits. The generated repulsion vectors are transferred to robot's joints if the robot is controlled at joints level, or transferred to the decoupled unit mass of the end-effector if the robot is controlled in operational space.

3.10.2 Joint limits

The control algorithm shall be capable of avoiding joints limits. In [50] and [2], collision avoidance was treated as an optimization problem, and their way of addressing joints-limits-avoidance (JLA) was through adding extra constraints to the optimization problem. Those constraints are inequalities that represent the limits imposed on joints variables. While in [39] the author addressed this issue by adding potential field, that is a function of joints angles, this potential is activated when the joint's angle is near its limit. From equations (28) and (29) in [39], and by noticing that the derivative of the potential field due to joint limit corresponds to repulsion torque, joints limits are addressed by adding repulsion torques that act on robot's joints when the configuration of the joint comes close, by some safety margin, to its limit. The repulsion torque can be chosen as reversely proportional to the error between the joint limit and the actual

joint position. In this study the following function was used to avoid joints limits:

$$\boldsymbol{\tau_{i,lim}} = \begin{cases} sign\left(q_i - q_{i,ll}\right) \frac{k}{\left(abs\left(q_i - q_{i,ll}\right)\right)^n}, & q_{i,ll} + \triangle q_i > q_i > q_{i,ll} \\ sign\left(q_i - q_{i,ul}\right) \frac{k}{\left(abs\left(q_i - q_{i,ul}\right)\right)^n}, & q_{i,ul} > q_i > q_{i,ul} - \triangle q_i \\ 0 & q_{i,ul} - \triangle q_i > q_i > q_{i,ll} + \triangle q_i \end{cases} \tag{3.72}$$

Where $\boldsymbol{\tau_{i,lim}}$ is the torque acting on joint i to avoid violating joint limit, $sign$ is a function that returns the sign of the operand, k is the proportionality constant, q_i is the position of joint i, $q_{i,ul}$ is the upper limit position of joint i, $q_{i,ll}$ is the lower limit position of that joint, $\triangle q_i$ is a safety margin, and n is the exponent. k and n can be adapted to get a better response.

3.10.3 Joint limits in the linearised dynamics approach

To incorporate JLA in our linearized-dynamics (3.71), the equation can be modified by adding repulsion torques:

$$\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}}_{\boldsymbol{e.att}} + \mathbf{J}_{\boldsymbol{e}}\mathbf{A}^{-\mathbf{1}}\left(\mathbf{J}_{\mathbf{cp}}^{\mathbf{T}}\boldsymbol{f_{rep}} + \boldsymbol{\tau_{lim}}\right) \triangle t \tag{3.73}$$

Where $\boldsymbol{\tau_{lim}}$ is the vector of repulsion torques that act on the joint when their angle come closer to its limit. The aforementioned controller was used through this study. However, after performing several tests, we conclude that the previous strategy is not optimal for our application, collision avoidance for collaborative robotics. In Figure 3.18 the external repulsion force due to the obstacle and the repulsion torque due to JLA are acting against each other. Thus, the repulsion vectors due to the coworker, and the repulsion torque due to joint limit avoidance, increases dramatically, while each element is trying to counteract the effect of the other. In such case we got oscillations, while in other scenarios the control failed to avoid the collision or failed to avoid the joint limit. To solve this problem we propose that when joint i come closer to its limit, we smoothly scale-down the action of that joint. In this study a scaling factor a_i was used:

$$a_i = \begin{cases} 0.5\left(1 - cos\left(\pi \frac{q_i - q_{i,ll}}{\triangle q_i}\right)\right), & q_{i,ll} + \triangle q_i > q_i > q_{i,ll} \\ 0.5\left(1 - cos\left(\pi \frac{q_i - q_{i,ul}}{\triangle q_i}\right)\right), & q_{i,ul} > q_i > q_{i,ul} - \triangle q_i \\ 1 & q_{i,ul} - \triangle q_i > q_i > q_{i,ll} + \triangle q_i \end{cases} \tag{3.74}$$

Where a_i is a unitless scaling factor associated with joint i. Other variables are the same as in (3.72). $\triangle q_i$ can be optimized according to the maximum angular acceleration applicable on joint i. The curve representing (3.74) is shown in Figure 3.19.

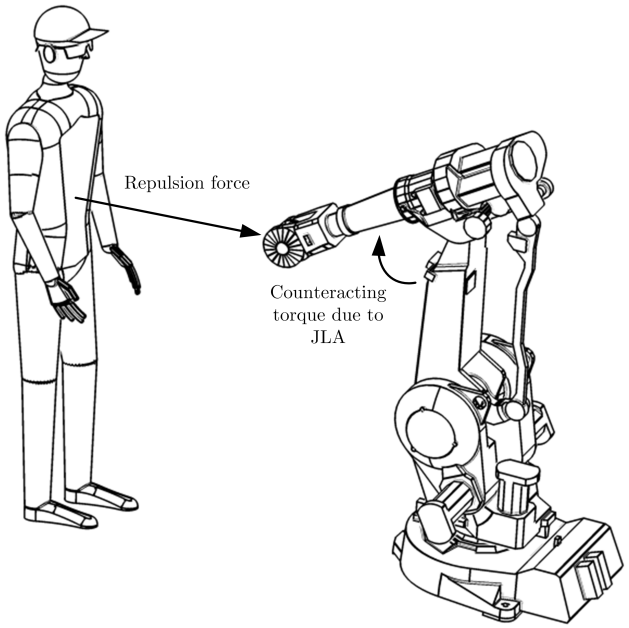

Figure 3.18: Drawback of repulsion torques method for incorporating joints limits

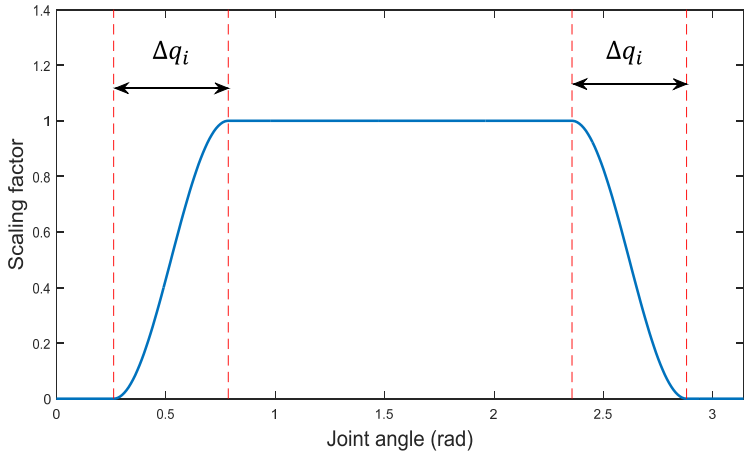

Figure 3.19: A curve representing the scaling factor a_i

To integrate our solution of joint limits avoidance into the linearized-dynamics controller (3.71), we collect the scaling factors a_i into a diagonal matrix $\mathbf{N}$:

$$\mathbf{N} = diag(\boldsymbol{a}) \tag{3.75}$$

Then, the modified control law of the linearized dynamics controller is:

$$\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}}_{\boldsymbol{e.att}} + \mathbf{J}_{\boldsymbol{e}}\mathbf{N}\mathbf{A}^{-\mathbf{1}}\mathbf{J}_{\mathbf{cp}}^{\mathbf{T}}\boldsymbol{f}_{\boldsymbol{rep}}\triangle t \tag{3.76}$$

This equation is a complete frame work for performing collision avoidance for non-redundant manipulators, it is formulated in operational space, it incorporates JLA, and the resulting motion takes into consideration robot's dynamics. Hence, it was used during the simulations of this study as the F controller.

3.10.4 Self-collision avoidance

Avoiding self-collisions is important in redundant manipulators and dual arm manipulators. For a 6 DOF serially linked robot self-collisions cannot occur if joints limits and work space limits were respected. Thus, when dealing with redundant manipulators susceptibility to self-collisions shall be taken into consideration. The procedure of avoiding self-collisions is similar to the collision avoidance with foreign obstacles, i.e., by calculating the minimum distance internally between robot's links that are likely to collide with each other. Afterwards, a repulsion vector is added, and transferred into an appropriate control command. This topic is studied in [61], a study entirely dedicated to self-collision avoidance.

3.11 Modifying attraction force to reduce the risk of collision

Since that this study is devoted to collision avoidance for collaborative human-robot interaction, the highest priority shall be dedicated to maintaining the safety of the human coworker under all conditions. A strategy for this purpose was introduced in (6.13) of [73], where a weighting factor c_v was introduced, and used to scale down the velocity of the end-effector towards the goal when an obstacle is nearby, thus reducing the risk of collision. In [73], c_v was defined as a combination of two cosine functions, one takes into consideration the direction of the repulsive force, while the other is based on its magnitude.

Following [73], in this section we describe a similar strategy to enhance the safety of the system and to reduce the risk of collision. Unlike [73], we sought this objective in a much simple way. To assure collision avoidance precedence over all other tasks, we modify the function of the attraction vector, where the attraction vector magnitude is scaled down the closer the coworker is to the robot. This is achieved by modifying the magnitude of the attraction vector using the following function:

$$\boldsymbol{v_{att.mod}} = \boldsymbol{v_{att}} \left(\frac{2}{1 + e^{-\left(\frac{d_{min}}{d_0}\right)^n}} - 1 \right) \tag{3.77}$$

Where $\boldsymbol{v_{att.mod}}$ is the modified attraction vector, $\boldsymbol{v_{att}}$ is the attraction vector from section 3.8.2, d_{min} is the minimum distance to the obstacle, d_0 is the distance at which the repulsion vector starts to act, and n is the exponent. The curve of the proposed modification coefficient is shown in Figure 3.20. Though simple in formulation, (3.77) demonstrated its robustness in the virtual-reality simulations performed throughout this study.

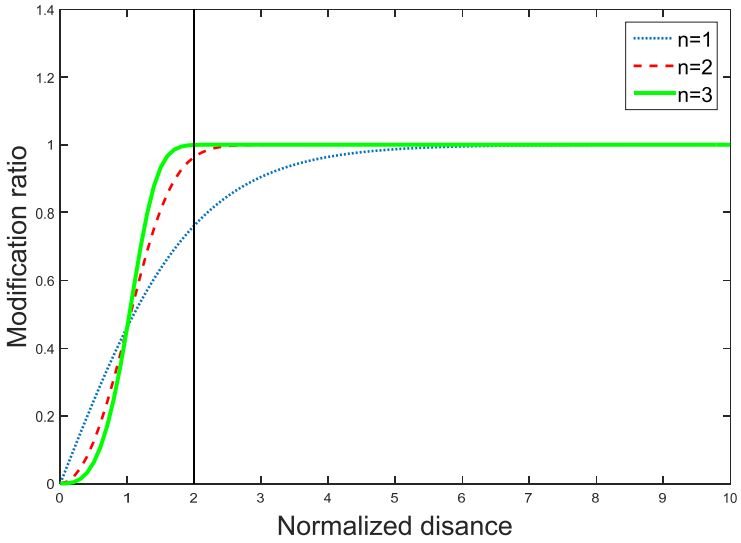

Figure 3.20: Modified of attraction force as a function of normalized distance

3.12 Drawbacks associated with artificial potential field

Despite the fact that the artificial potential field is a powerful method for performing collision avoidance, it can be improved. Some problems associated with its implementation have been reported. In [78] the authors identified five drawbacks that pertains to path planning for mobile robots using the potential field method. In this study, similar problems, but in the context of robotic manipulators, were noticed:

1. Oscillations when the robot is subjected to several counteracting, repulsion forces from several obstacles.

2. Goal Non Reachable Obstacles Near By (GNRON) as described in [78].

The above problems where noticed while performing tests, we describe them in the following.

3.12.1 Oscillations

Oscillations occur when the robot is confined in spaces of reduced manipulability, i.e. when it is subjected to repulsive forces from different potential sources. To solve this problem a moving average filter was utilized to filter the signal before sending it to the robot. Despite their simplicity, moving average filters are optimal and offer a solution for reducing random noise and smoothing out rapid changes in a signal. The mathematical representation of the moving average filter is:

$$v_i = \frac{1}{m} \sum_{j=0}^{m-1} x_{i-j} \tag{3.78}$$

Where v_i represents the filtered signal, x_{i-j} is the original signal, and m is the number of samples used for averaging.

3.12.2 Goal Non-Reachable Obstacles Nearby (GNRON)

This problem shows up when the obstacle and the goal are close to each other's, especially when the obstacle is wedged between the robot and the goal. This problem was described in [78]. However, the discussion was in the context of path planning for mobile robots. This problem was also identified during simulations performed in the context of this study. Figure 3.21 demonstrates this problem. In the figure $\boldsymbol{O}$ represents obstacle's center and $\boldsymbol{T}$ represents goal position. In such situation the robot was trapped in an oscillatory motion and the direction of oscillations is shown in a double-pointed arrow. In [78] a solution for this problem was presented, through modifying the function of the attraction potential field. Nevertheless the solution presented was kept in the framework of path-planning for mobile robots, which is not directly applicable for robotic manipulators. In Chapter 4 of this study testing results associated with this problem are presented.

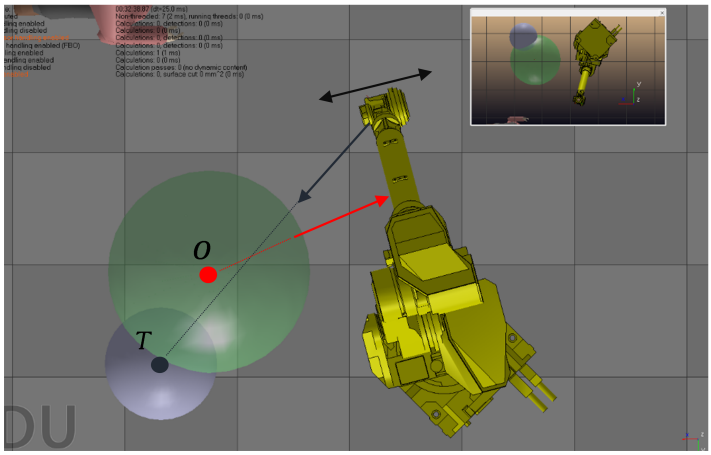

Figure 3.21: Goals non-reachable with obstacle nearby

Chapter 4

Experiments and Results

In the context of the methodology presented in this study and to validate the proposed framework several real-time virtual-reality simulations were performed. In these simulations the robot used is 6 DOF serially linked manipulator. The robot is idle or performing a task, while an obstacle is moving in a collision course with the robot. Statistical data of velocities and positions were collected, and the results were analyzed. The simulations were run on Intel® Core i7 machine, with 4 GB of RAM. The overall simulation update time was 0.158 seconds/cycle on average.

4.1 Testing set and Results

Tests 1, Test 2 and Test 3 were performed to measure the performance of the proposed kinematics controller under different configurations. Test 4 was performed to evaluate the performance of the force controller. In Test 5 the kinematics controller is implemented in a typical robotic cell. In this cell the robot is performing a pick and place operation while avoiding collision with a coworker moving randomly in the shared workspace. Test 6 was performed to demonstrate the problem of GNRON and Test 7 measures the effectiveness of scaling factor method for achieving JLA. Table 4.1 shows the parameters for each test.

Table 4.1: Tests parameters (K-kinematics controller, F-force controller).

Test	Distance of influence m	Controller	Approaching direction	MVA filter
1	1	K	$X-direction$	***yes/no***
2	0.5	K	$X-direction$	***yes***
3	0.7	K	$Y-direction$	***yes***
4	1	F	$X-direction$	***yes***
5	1	K	*Random*	***yes***
6	1	K	*Random*	***yes***
7	1	K	*Random*	***yes***

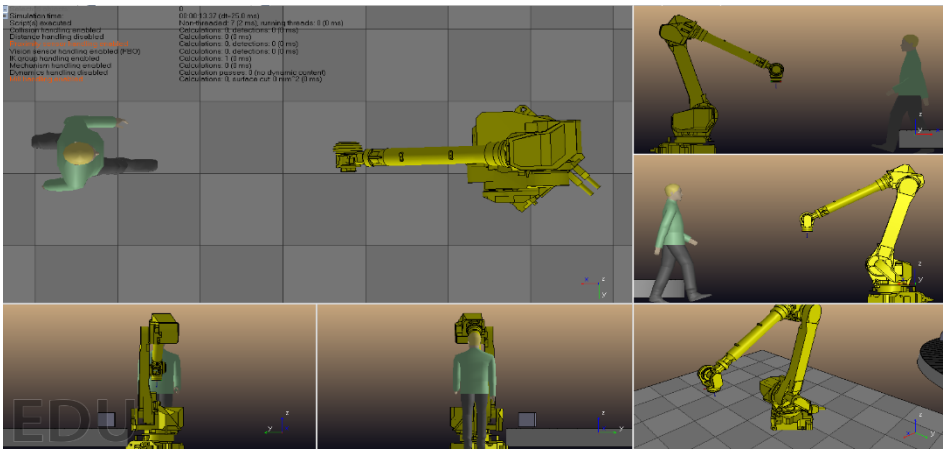

Figure 4.1: Layout of simulation scene for Test 1

4.1.1 Test 1

In this test the robot is stationary in resting position and the human coworker is approaching the robot from the front (along X direction), Figure 4.1. The coworker walks towards the end-effector, the minimum distance between them decreases, and when it reaches a predefined safety-distance the robot reacts by moving-away to avoid collision with the coworker. After performing the real-time virtual reality simulation of the aforementioned scene, it can be noticed that the robot manages to avoid collision with the coworker successfully. To quantify the response of the robot, several parameters where collected during the simulation:

1. The position of the end-effector expressed in Cartesian coordinates of the reference frame;
2. The position of a reference point on the coworker expressed in Cartesian coordinates of the reference frame;
3. Minimum distance between the coworker and the robot[1];
4. Time of each data sample;
5. Velocity of the end-effector;
6. Velocity of a reference point on the coworker;

The response of the robot is analyzed using the plot in Figure 4.2 that describes the X coordinate of the end-effector, the X coordinate of the coworker and the difference between them.

[1] Calculated from the formulas presented in section 3.6

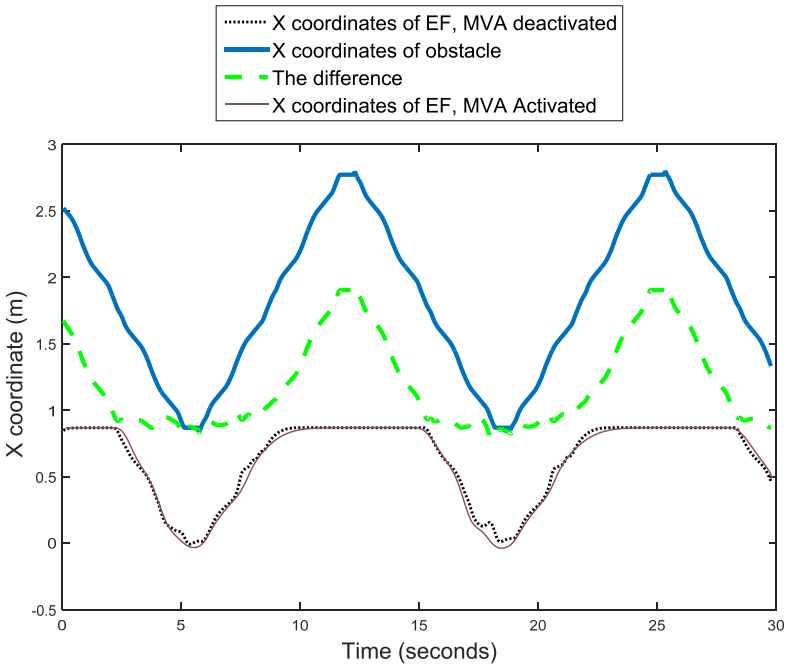

Figure 4.2: Test 1 results.

We can notice from the plot that at the beginning of the simulation the robot is stationary in a resting position, while the coworker is moving with a negative velocity about 0.3 m/sec, along X direction towards the robot. When a predefined safe distance between the robot and the coworker is reached, the robot reacts by moving away from the coworker. When the coworker changes its motion to the opposite direction, the robot moves back towards its resting position.The simulation cycle continue for several more periods and the same reaction was noticed in each period. To show the influence of moving average filter (MVA) on reducing vibrations, we plot the response for the same simulation but in the scenario where MVA is deactivated. It is clear that the moving average filter was successful in suppressing the vibrations.

Figure 4.3 shows the evolution of the minimum distance between the coworker and the robot as function of time. Data pertaining to several simulation cycles were collected. The minimum distance reported is 0.75 meter.

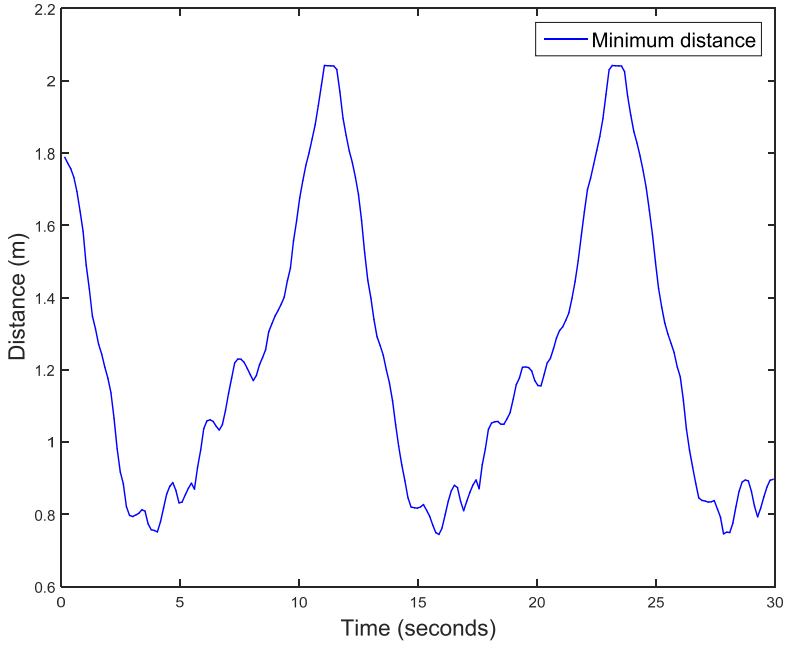

Figure 4.3: Minimum distance between robot and cowrker for Test 1.

Figure 4.4 shows the X component of end-effector's velocity in two situations: using the MVA filter and without using the MVA filter. The moving average filter was successful in suppressing the vibrations and achieving a smoother reaction from the robot.

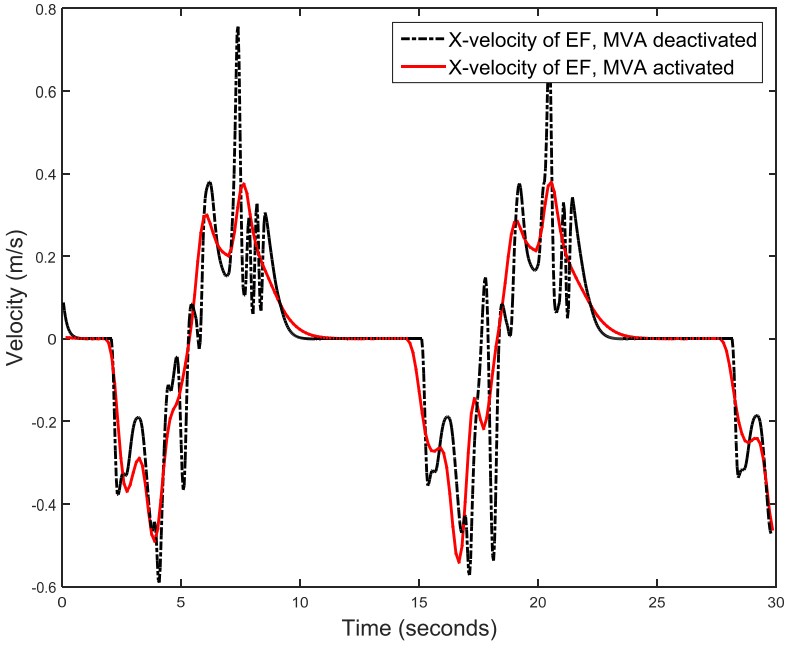

Figure 4.4: End-effector velocity for Test 1.

4.1.2 Test 2

This test is similar to Test 1. The only difference is in the dimension of the area of influence[2] around the obstacle, which is changed from 1 meter to 0.5 meter. From Figure 4.5 we see that reducing the area of influence caused the minimum difference of X coordinate between the end-effector to the coworker to change from 0.82 meter for Test 1, to 0.32 for Test 2. And Figure 4.6 shows that the reduction of the area of influence caused the minimum distance to change from 0.75 m for Test 1, to 0.35 m for Test 2. Figure 4.7 show the X component of the end-effector's velocity as function of time, where the maximum reaction velocity reached is 0.41 m/sec as opposed to 0.56 m/sec for Test 1. In addition, velocity profile for Test 2 appears to be smoother.

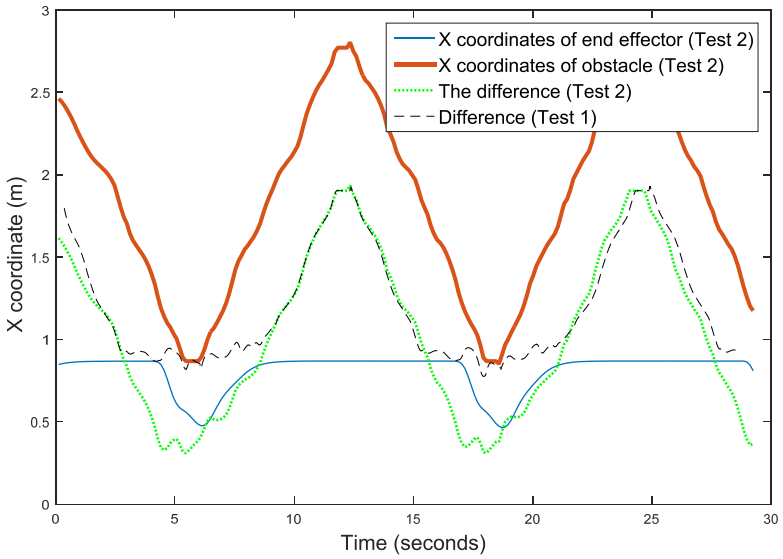

Figure 4.5: Test 2 results.

[2]During the tests presented in this section the dimension of the area of influence is calculated as the minimum distance between obstacle's primitive center and the robot's primitive center, i.e. in the case of two cylinders, the measure of the area of influence is considered as the minimum distance between their corresponding axis segments.

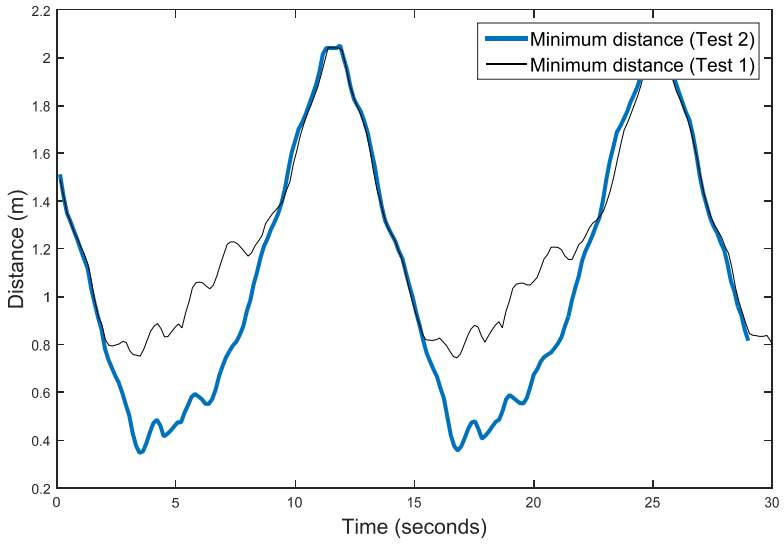

Figure 4.6: Minimum distance between coworker and robot.

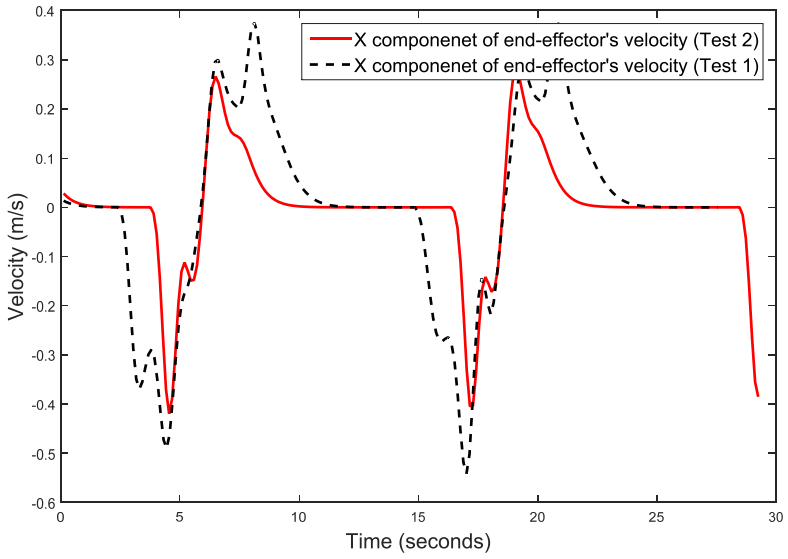

Figure 4.7: End-effector's velocity.

4.1.3 Test 3

In this test the coworker approaches the robot from the side along Y direction, Figure 4.8. At the beginning the coworker is 2.5 meters away from the end-effector and then he/she starts walking toward the robot, with an average velocity of 0.2 m/sec in the Y direction. Figure 4.9 shows that at the beginning the end-effector is stationary while the coworker is walking towards the robot. By the time the safety distance between the robot and the coworker, 0.7 meter, is reached the robot starts to react to avoid the collision with the coworker. From the curve in Figure 4.9 we notice that the minimum Y distance reached between the end-effector and the coworker is 0.52 meters. This approach by the side is different from the frontal approach in terms of robot's reaction, in which collision avoidance relies mainly on motion along axis 1 of the robot. Nevertheless, the results obtained are similar. Figure 4.10 shows the evolution of the minimum distance between the robot and the obstacle, in which the minimum distance reached is 0.5 meters, in line with previous tests.

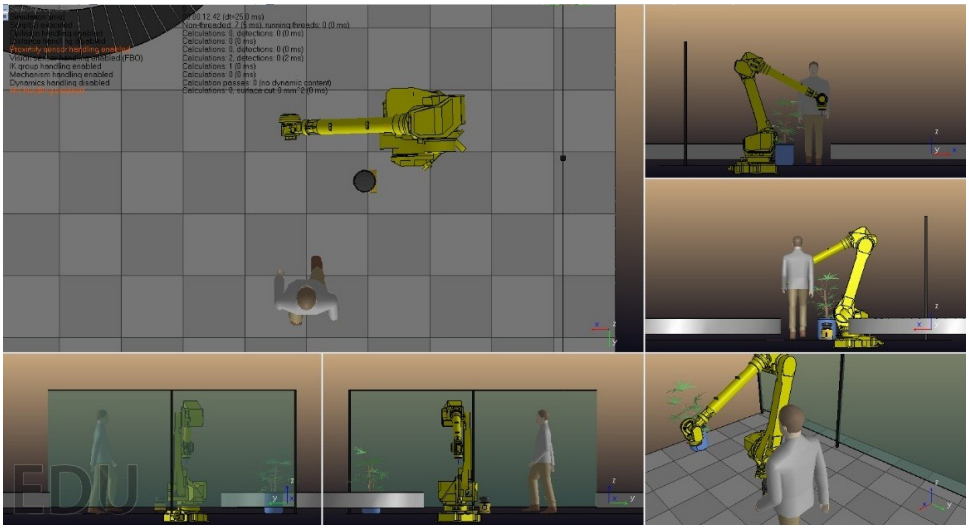

Figure 4.8: Test 5 simulation scene.

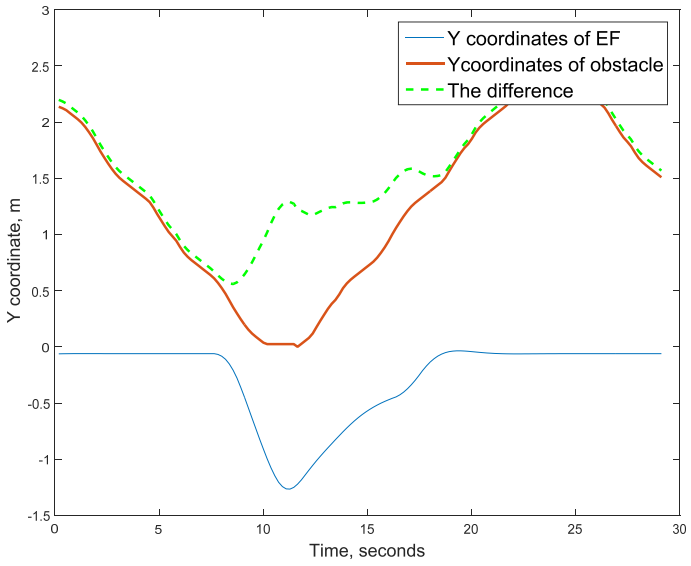

Figure 4.9: Test 3 results.

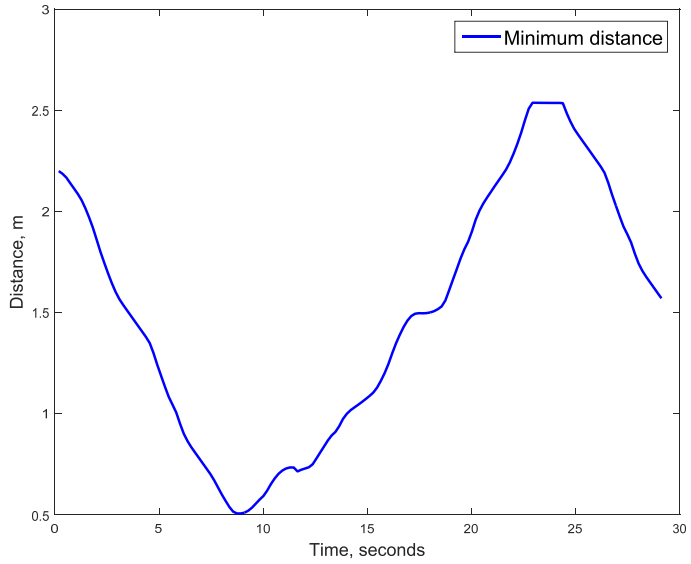

Figure 4.10: Test 3 minimum distance between robot and coworker.

4.1.4 Test 4

While in the previous simulations the kinematics controller was used to test the system, in Test 4 the force control strategy, described in 4.3.2, is applied. The simulation scene of this test is identical to the scene of Test 1.

Figure 4.11 shows that the minimum value of the distance difference in the X direction is 0.7m as opposed to 0.75m in Test 1. Figure 4.12 shows that the maximum reaction velocity is 0.61 m/sec as opposed to 0.56 m/sec attained in Test 1.

Using the F controller resulted in a different reaction motion when compared to the reaction motion generated using the K controller. This is because the F controller takes into consideration robot's dynamics for generating the motion. The presence of the inverse of the inertia matrix causes the joints associated with the lowest articulated inertia to have the highest rate, in other words, the F controller generates faster motions to joints associated with lowest inertias. The result is a more natural way for motion generation.

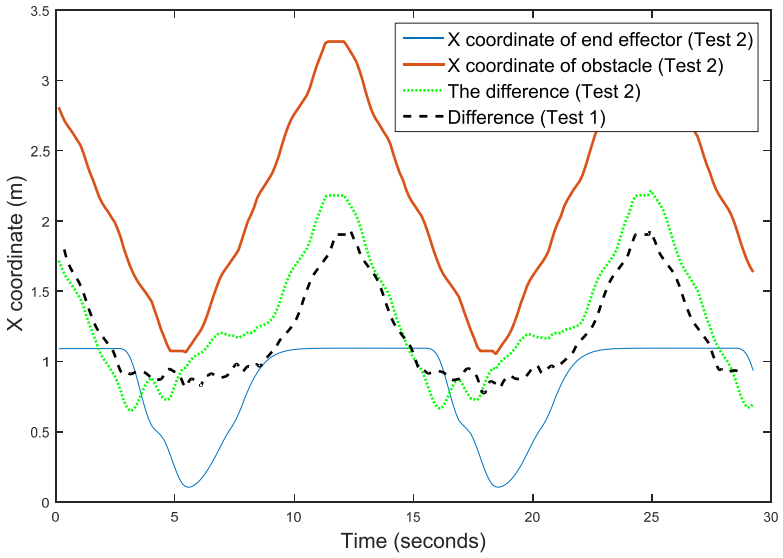

Figure 4.11: Test4 results.

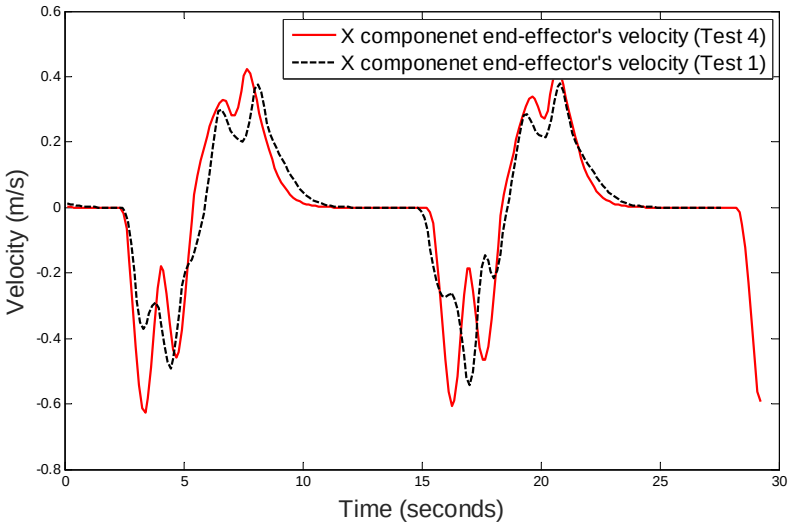

Figure 4.12: End-effector's velocity.

4.1.5 Test 5

In test 5 the robot is manipulating an object from one conveyor to another. It is required that the robot moves as close as possible to a predetermined nominal trajectory, while avoiding collision with human coworker that shares the same workspace. For performing this test a state-flow control logic was developed based on state machine diagram in Figure 4.13. At any time the simulation progresses in one of seven different states which are:

1. In the first step the robot moves toward the home position.
2. Once the home position is reached, the robot moves along a predefined trajectory towards the object.
3. Once the object is reached the robot picks the object.
4. Then the robot moves along a predefined trajectory towards the goal position.
5. Once reached, the robot places the object at the goal position.
6. The robot moves back to home position.
7. At any state, whenever the coworker is closer than some safety distance from the robot, the state will transit to collision avoidance mode, and the collision-avoidance controller is activated.

Figure (4.14) shows motion trajectories of the robot and the coworker. Figure (4.15) shows snapshots of the simulation progression, the simulation was performed successfully, the robot avoided collisions with the coworker and the objectives of the method were satisfied.

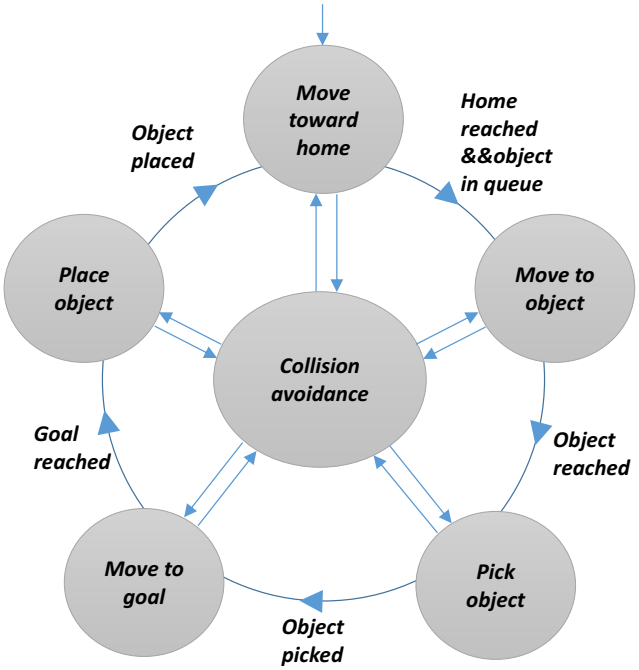

Figure 4.13: State machine for robotic cell control with collision avoidance capability.

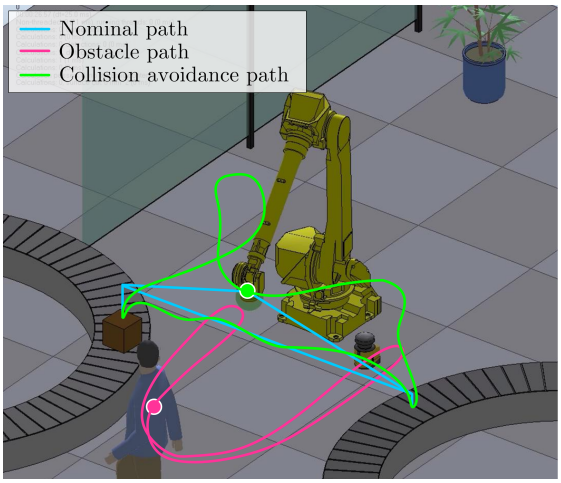

Figure 4.14: Motion trajectories.

Figure 4.15: Robotic cell simulation.

4.1.6 Test 6

This test was performed to demonstrate the problem of GNRON. Figure 4.16 shows the simulation scene. In such situation the robot was trapped in an oscillatory motion, the direction of oscillations are shown by the double-pointed arrow in the figure.

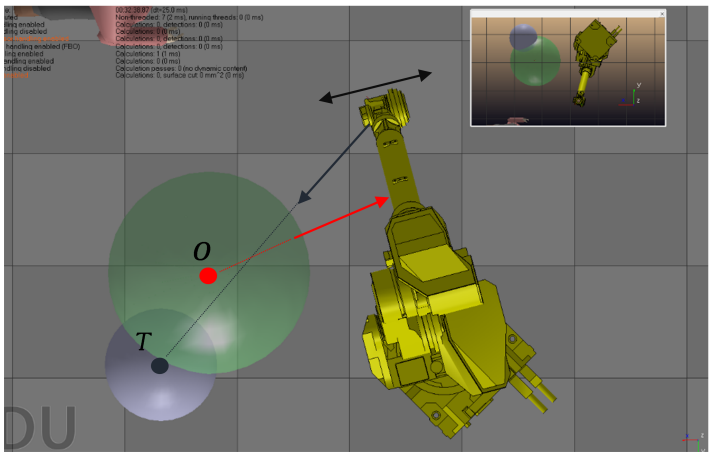

Figure 4.16: Goal non-reachable with obstacle nearby.

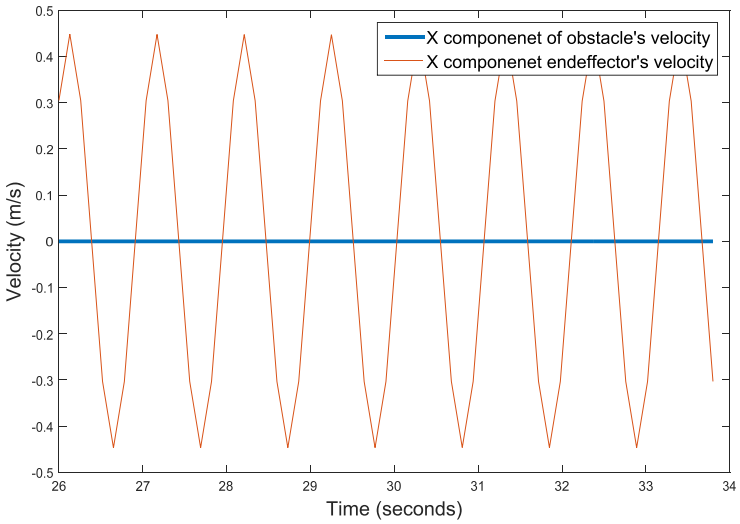

Figure 4.17: Test 6 velocity results.

From Figure 4.16, statistical data pertaining to the end-effector and the obstacle's were collected. Figure 4.17 shows the X component of end-effector's velocity and the X component of obstacle's velocity. The obstacle is stationary while the end-effector is oscillating. The oscillatory phenomena of the end-effector is evident in the X coordinate of the end-effector in Figure 4.18, while the obstacle is stationary as represented by

the horizontal line.

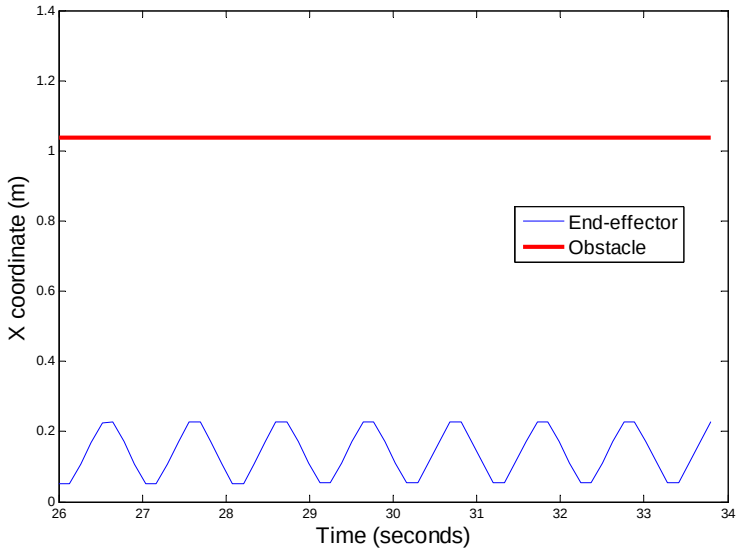

Figure 4.18: Test 6 results.

4.1.7 Test 7

Test 7 is performed to evaluate the performance of the scaling factor method as opposed to assigning a potential field to the joint, for achieving JLA. Thus, three simulations are performed. In the first and second simulations a potential field was added to each joint. This potential field is activated when the joint is near its limit, causing it avoid its limit. In the third simulation the scaling factor was utilized, as defined in (3.74).

For the three simulations, the scene proposed is illustrated in Figure 4.19. In the beginning of the simulation the robot and the coworker are stationary as illustrated in subfigure (a). Then, the human coworker walks towards the end-effector from the front, as a result the minimum distance between the robot and coworker decreases, until it reaches some predefined safety distance, 0.7 meter for this simulation, at which the robot starts to react, subfigure (b). Then, the coworker continues its motion towards the robot until it is about 25 cm from the robot, subfigure (c). Our choice of placing the coworker in such a close proximity to the manipulator is to force the third joint to reach its minimum limit, so that we can test the performance of JLA algorithm. In all three simulations the minimum limit chosen for the joint is 10 degrees. The first and second simulations are totally identical but they differ in the strength of the potential field utilized. To assess the performance of the algorithms, the following data are acquired:

1. Third joint angle, to measure the algorithm's performance in avoiding joints limits.

2. X component of end-effector's velocity.

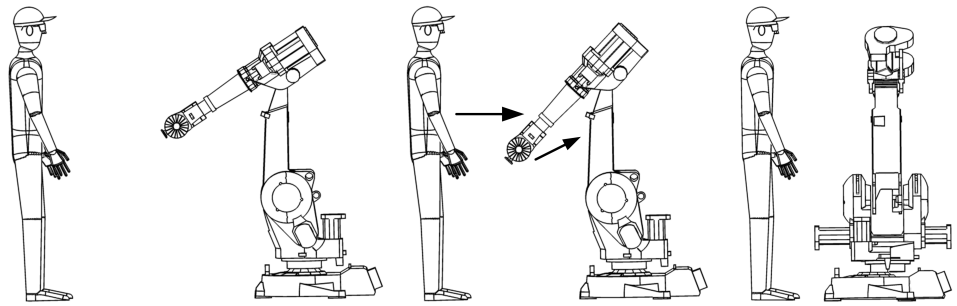

Figure 4.19: Scene overview.

4.1.7.1 Simulation 1

In this simulation a potential field is used to incorporate JLA. Figure 4.20 shows the resulting angle variations of third joint as a function of time. In the figure we see that the third joint failed to avoid its limit because the strength of the potential field chosen is not satisfactory.

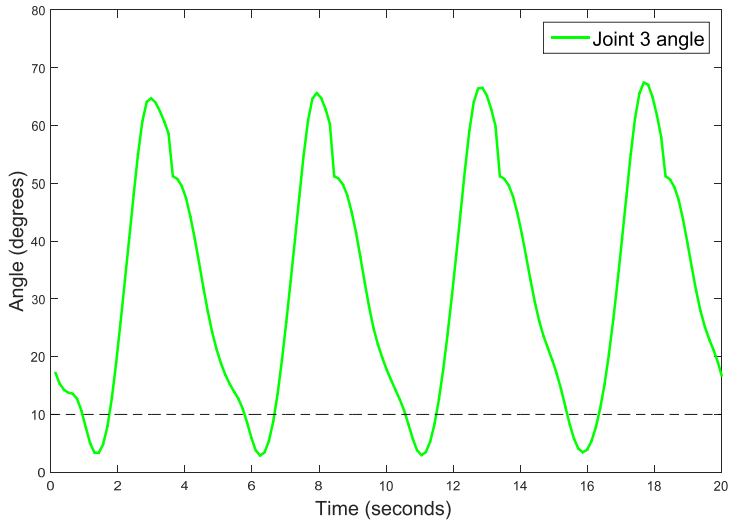

Figure 4.20: Third joint angle for simulation 1.

4.1.7.2 Simulation 2

To achieve limit avoidance of joint three, the strength of the potential field was increased. Figure 4.21 shows the joint's angle as function of time. It can be seen that after increasing the strength of the potential field the joint is able to avoid its limit. Figure 4.22 shows the velocity of the end-effector where the jerky sections of the curve are highlighted by the green frames.

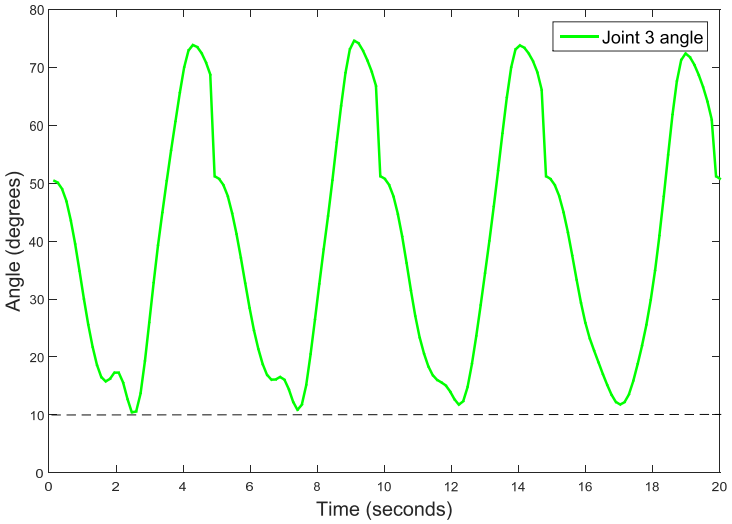

Figure 4.21: Third joint angle for simulation 2.

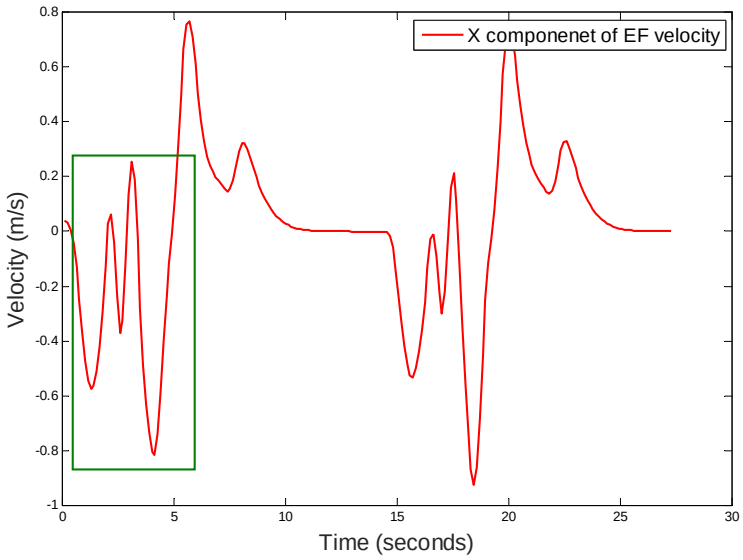

Figure 4.22: End-effector X component of velocity.

4.1.7.3 Simulation 3

In this simulation the joint limit avoidance strategy is changed. The scaling factor method was utilized instead of the potential field. Figure 4.23 shows that the scaling factor strategy was successful in achieving joint limits avoidance. Even better, the minimum value reported for joint three was 12.4 degrees, an improved performance over the previous simulation where the minimum value attained for joint three was close to 10 degrees. By comparing the reaction curves, especially the parts of the curves confined with the green frames with their associate from simulation 2, we notice that the scaling factor strategy gives a smoother reaction.

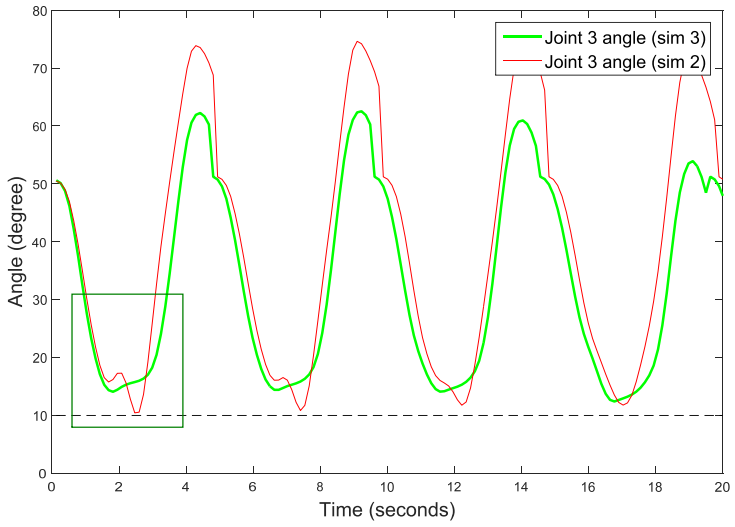

Figure 4.23: Third joint angle, scaling factor method.

Figure 4.24 shows the X-component of the velocity of the end-effector.From the areas of the curve highlighted with green frames we notice that the reaction attained in the scaling factor method is smoother than the reaction attained when the potential feild strategy was utilized.

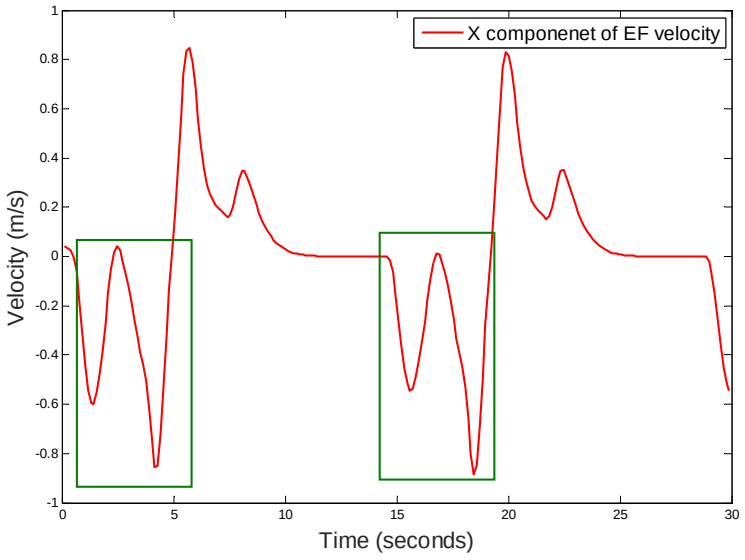

Figure 4.24: End-effector X component of velocity.

4.2 Discussion

From the previous simulations we see that in general the proposed collision avoidance algorithm achieved its requirements while:

- Test 1 shows that the K controller with MVA filter combined with the scaling factor method for achieving JLA attains remarkable results. This controller was tested with different parameters and under different configurations in Test 2, Test 3 and Test 5. The results were satisfactory.
- In Test 4 the force controller achieved smooth collision avoidance motion. Unlike the Kinematics controller, the dynamics controller generated higher angular velocities in the directions associated with lower articulated inertias and smaller angular velocities for joints associated with higher articulated inertias. However, due to time limitations we were unable to provide a full analysis of this controller.
- Scaling down the attraction vector magnitude when the obstacle is near the robot assured better safety for the system.
- As demonstrated by Test 6 the scaling factor method demonstrated superior performance for dealing with JLA, utilizing this method attains safer reactions, because it assures zero velocity for joints approaching their limits in situations where there are conflicts between avoiding collision and avoiding joint limits.
- The proposed system allows contact avoidance and provide a safety margin that can be adjusted by changing the distance of the area of influence around the obstacle.
- The proposed system can be easily integrated into a higher level control architecture, as demonstrated in Test 5, for assuring safe human-robot collaboration even for situations where the robot is performing complex tasks.
- The system behaves well regardless of the direction of approach, front, side or randomly.
- The system is computationally efficient, it is easy to implement, and it can be adapted easily to operate on different types of robots using their specific parameters (Denavit Hartenberg, links masses and inertias).
- The system suffers from GNRON drawback as demonstrated in Test 7, solving this problem will be left for future work.

Chapter 5

Conclusion and Future work

5.1 Conclusion

In this study we proposed GDA, a novel algorithm for representing the dynamics of serially linked robots, and for calculating its joint space inertia matrix, joint space Coriolis matrix, joint space Centrifugal matrix, and the time derivative of joint space inertia matrix. In-addition we presented GDAHJ an algorithm that achieves better efficiency over state-of-the-art when calculating JSIM for hyper-joint manipulators. This increase in efficiency is achieved through minimizing the number of operations that has $O(n^2)$ computational complexity. In such case, the number of computations associated with the quadratic terms are reduced to the minimum value possible, from $(16n^2)$ in the case of CRBA to $(5.5n^2)$ for the proposed algorithm. In addition we described and proved the frame injection principal. The proposed algorithms are implemented in MATLAB®, six functions were realized for calculating the dynamics matrices efficiently. In addition comparison between the proposed algorithm against other well established algorithms was made, the performance of the proposed algorithm was discussed in operation count section of this study.

The computational efficiency of the realized MATLAB® functions compares favourably with existing code. After comprehensive search on MATLAB® robotics toolboxes, according to our knowledge, the implemented GDA code, is the most efficient MATLAB® code available for calculating Centrifugal and Coriolis matrices numerically.

We perceive that GDA's way of representing robot dynamics in a mathematical form resembling the equation of the inverse dynamics as intuitive, simple and easy to implement. It's efficiency is reflected by the $O(n^2)$ algorithm deduced for calculating the time derivative of joint space inertia matrix. By using this methodology other parameters of robot dynamics can be calculated efficiently in a like manner to what had been presented in this study.

Also we presented a novel real-time collision avoidance algorithm based on the artificial potential fields, i.e., using attraction and repulsion vectors. The control process is based on two main principals: (1) generation of end-effector's motion close as possible to a predefined trajectory, and (2) generation of collision avoidance motion from an estimated collision imminence measure. To estimate collision imminence, the min-

imum distance between the robot and the moving obstacle is used. The geometry of the robot and the coworker are approximated using primitive geometries, i.e. they are modelled using cylinders and spheres. After modelling, a methodology for measuring the distance between the robot and the obstacle is developed. A novel method for calculating the minimum distance between cylinders is proposed. The proposed method reformulates minimum distance calculation into an optimization problem constrained by bounded value variables. For attaining a solution, a novel method utilizing QR factorization and coordinate transformation is proposed. Based on this method an efficient formulation for collision detection between cylinders is proposed.

A repulsion vector is calculated based on the minimum distance between the robot and the obstacle and the attraction vector is calculated based on the error vector between the end-effector and required goal position. Those vectors are used by the controller to control the robot. Two controllers with collision avoidance capability are presented. The kinematics controller (K controller) is a kinematics-level controller which utilizes inverse-kinematics calculations. Two different methods for inverse-kinematics computation are utilized, the damped least squares and the Jacobean transpose. The second controller (F controller), is force based controller that takes into consideration robots dynamics for generating the motion. The implementation was successful, though during early simulations three main issues were identified, a solution is proposed to solve those issues and a system with better response is achieved. The vibrations problem appeared when the robot is in areas of reduced manipulability, subjected to several forces. This problem was solved by utilizing moving average filter. Failure to avoid joint limits happened when the obstacle is in close proximity to the robot, where the repulsion vector and the action due to joint limit are counteracting each other and the concept of scaling factor was utilized to solve the problem. The drawback of GNRON appeared when the obstacle is wedged between the robot and the goal, due to time limitation this problem remains unresolved, and it is open for future work.

The study is concluded with real-time simulations for testing the proposed algorithms. The main computations are implemented in MATLAB®, while real-time virtual-reality simulations are carried out in V-REP, the two programs are connected together through sockets. Several statistical data were attained from simulations. Those data pertains to the robot's dynamical response and the distance between the robot and the coworker.

Then the collision avoidance controller is embedded in a higher level hybrid automata control system that is developed to control a typical industrial robotic cell. The industrial cell was simulated successfully, i.e. the robot was able to avoid collision with the coworker while achieving the required objective.

In addition to the tests presented in this study, other tests were performed, where the obstacle is moved in random directions and the response of the robot is observed. The system gave good performance, and we concluded that the final system is robust and able to achieve collision avoidance for collaborative robotics. Based on this confidence, we envisage implementing the system on real robot.

5.2 Future work

Future work will focus on implementing the collision avoidance method in this study on a real robotic manipulator, the system will integrate Kinect sensor for detecting obstacles and their position, this data will be provided to the proposed algorithm and the output will be used to command the manipulator. In addition extra efforts will be dedicated for solving the problem of GNRON for achieving a robust system.

Also, the author would like to extend the collision avoidance method presented in this study for mobile robotic manipulators applications. The final goal is to create an advanced manipulation system, based on the idea of attraction and repulsion vectors, in which a robotic manipulator mounted on a mobile platform is able to navigate autonomously in a cluttered environment, avoid collisions with obstacles and reach the designated goal. In this system the operator commands the robot to grasp an object, the mobile platform navigates towards the object, and the mounted arm picks the required object and deliver it back to the operator. The key for performing the control for this system will be the concept of attraction and repulsion vectors presented in this study.

Bibliography

[1] Knut B Kaldestad, Sami Haddadin, Rico Belder, Geir Hovland, David Anisi, et al. Collision avoidance with potential fields based on parallel processing of 3d-point cloud data on the gpu. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3250–3257. IEEE, 2014.

[2] Paul Bosscher and Daniel Hedman. Real-time collision avoidance algorithm for robotic manipulators. *Industrial Robot: An International Journal*, 38(2):186–197, 2011.

[3] Knut Berg Kaldestad. Industrial robot collision handling in harsh environments. 2014.

[4] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. Springer Science & Business Media, 2009.

[5] Roy Featherstone and David Orin. Robot dynamics: equations and algorithms. In *ICRA*, pages 826–834, 2000.

[6] CA Balafoutis. A survey of efficient computational methods for manipulator inverse dynamics. *Journal of Intelligent and Robotic Systems*, 9(1-2):45–71, 1994.

[7] Oussama Khatib. Dynamic control of manipulator in operational space. In *Proc. 6th IFToMM World Congress on Theory of Machines and Mechanisms*, pages 1128–1131, 1983.

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

[9] Kyong-Sok Chang. *Efficient algorithms for articulated branching mechanisms: dynamic modeling, control, and simulation*. PhD thesis, Citeseer, 2000.

[10] Kyong-Sok Chang and Oussama Khatib. Efficient recursive algorithm for the operational space inertia matrix of branching mechanisms. *Advanced Robotics*, 14(8):703–715, 2001.

[11] John J Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson Prentice Hall Upper Saddle River, 2005.

[12] Wisama Khalil. Dynamic modeling of robots using recursive newton-euler techniques. In *ICINCO2010*, 2010.

[13] Peter Corke. *Robotics, vision and control: fundamental algorithms in MATLAB*, volume 73. Springer Science & Business Media, 2011.

[14] Mark W Spong. Control of robots and manipulators. *The control handbook*, pages 1339–1351, 1996.

[15] Lorenzo Sciavicco, Bruno Siciliano, and Luigi Villani. Lagrange and newton-euler dynamic modeling of a gear-driven robot manipulator with inclusion of motor inertia effects. *Advanced robotics*, 10(3):317–334, 1995.

[16] Metin Toz and Serdar Kucuk. Dynamics simulation toolbox for industrial robot manipulators. *Computer Applications in Engineering Education*, 18(2):319–330, 2010.

[17] Emmanuel Dean-Leon, Saurabh Nair, and Aaron Knoll. User friendly matlab-toolbox for symbolic robot dynamic modeling used for control design. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 2181–2188. IEEE, 2012.

[18] Charles P Neuman and John J Murray. Symbolically efficient formulations for computational robot dynamics. *Journal of robotic systems*, 4(6):743–769, 1987.

[19] Herman Hoifodt. Dynamic modeling and simulation of robot manipulators: the newton-euler formulation. 2011.

[20] Thomas R Kane and David A Levinson. The use of kane's dynamical equations in robotics. *The International Journal of Robotics Research*, 2(3):3–21, 1983.

[21] H. Baruh. *Analytical dynamics*. MacGraw-Hill, 1999.

[22] M Vukobratović, Shi-Gang Li, and N Kirćanski. An efficient procedure for generating dynamic manipulator models. *Robotica*, 3(03):147–152, 1985.

[23] John YS Luh, Michael W Walker, and Richard PC Paul. On-line computational scheme for mechanical manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102(2):69–76, 1980.

[24] 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.

[25] R. Featherstone. Spatial vectors and rigid-body dynamics.

[26] Roy Featherstone. Plucker basis vectors. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1892–1897. IEEE, 2006.

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

[28] Frank C. Park, James E Bobrow, and Scott R Ploen. A lie group formulation of robot dynamics. *The International Journal of Robotics Research*, 14(6):609–618, 1995.

[29] Scott R Ploen and Frank C Park. Coordinate-invariant algorithms for robot dynamics. *Robotics and Automation, IEEE Transactions on*, 15(6):1130–1135, 1999.

[30] Giovanni Legnani, Federico Casolo, Paolo Righettini, and Bruno Zappa. A homogeneous matrix approach to 3d kinematics and dynamicsi. theory. *Mechanism and machine theory*, 31(5):573–587, 1996.

[31] Giovanni Legnani, Federico Casalo, Paolo Righettini, and Bruno Zappa. A homogeneous matrix approach to 3d kinematics and dynamicsii. applications to chains of rigid bodies and serial manipulators. *Mechanism and machine theory*, 31(5):589–605, 1996.

[32] Alessandro De Luca and Lorenzo Ferrajoli. A modified newton-euler method for dynamic computations in robot fault detection and control. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3359–3364. IEEE, 2009.

[33] Alessandro De Luca, Alin Albu-Schaffer, Sami Haddadin, and Gerd Hirzinger. Collision detection and safe reaction with the dlr-iii lightweight manipulator arm. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 1623–1630. IEEE, 2006.

[34] Magnus Bjerkeng and Kristin Y Pettersen. A new coriolis matrix factorization. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4974–4979. IEEE, 2012.

[35] Martin Becke and Thomas Schlegl. Extended newton-euler based centrifugal/coriolis matrix factorization for geared serial robot manipulators with ideal joints. In *MECHATRONIKA, 2012 15th International Symposium*, pages 1–7. IEEE, 2012.

[36] Dario Bellicoso Marco Caputano. Damarob, dario and marco's robotics symbolic toolbox for matlab.

[37] Peter I Corke et al. A computer tool for simulation and analysis: the robotics toolbox for matlab. In *Proc. National Conf. Australian Robot Association*, pages 319–330, 1995.

[38] Roy Featherstone. Efficient factorization of the joint-space inertia matrix for branched kinematic trees. *The International Journal of Robotics Research*, 24(6):487–500, 2005.

[39] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[40] P Chotiprayanakul, DK Liu, D Wang, and G Dissanayake. A 3-dimensional force field method for robot collision avoidance in complex environments. 2007.

[41] Yonghong Bu and Stephen Cameron. Active motion planning and collision avoidance for redundant manipulators. In *Assembly and Task Planning, 1997. ISATP 97., 1997 IEEE International Symposium on*, pages 13–18. IEEE, 1997.

[42] Fabrizio Flacco, Torsten Kröger, Alessandro De Luca, and Oussama Khatib. A depth space approach to human-robot collision avoidance. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 338–345. IEEE, 2012.

[43] Sami Haddadin, Rico Belder, and Alin Albu-Schäffer. Dynamic motion planning for robots in partially unknown environments. In *IFAC World Congress (IFAC2011), Milano, Italy*, volume 18, 2011.

[44] Rabie Soukieh, Iman Shames, and Baris Fidan. Obstacle avoidance of nonholonomic unicycle robots based on fluid mechanical modeling. In *Control Conference (ECC), 2009 European*, pages 3269–3274. IEEE, 2009.

[45] Rainer Palm and Dimiter Driankov. Fluid mechanics for path planning and obstacle avoidance of mobile robots. In *Informatics in Control, Automation and Robotics (ICINCO), 2014 11th International Conference on*, volume 2, pages 231–238. IEEE, 2014.

[46] Ahmad Masoud, Samer Masoud, Mohamed M Bayoumi, et al. Robot navigation using a pressure generated mechanical stress field: the biharmonic potential approach. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 124–129. IEEE, 1994.

[47] T. Nishimura, K. Sugawara, I. Yoshihara, and K. Abe. A motion planning method for a hyper multi-joint manipulator using genetic algorithm. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, volume 4, pages 645–650 vol.4, 1999.

[48] Leon Žlajpah and Bojan Nemec. Kinematic control algorithms for on-line obstacle avoidance for redundant manipulators. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 2, pages 1898–1903. IEEE, 2002.

[49] L Žlajpah and T Petrič. Serial and parallel robot manipulators kinematics, dynamics, control and optimization, chap obstacle avoidance for redundant manipulators as control problem, 2012.

[50] S Mitsi, K-D Bouzakis, and G Mansour. Optimization of robot links motion in inverse kinematics solution considering collision avoidance and joint limits. *Mechanism and machine theory*, 30(5):653–663, 1995.

[51] Lydia Kavraki1 Petr Svestka and Jean-Claude Latombe1 Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces.

[52] Tobias Kunz, Ulrich Reiser, Mike Stilman, and Alexander Verl. Real-time path planning for a robot arm in changing environments. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 5906–5911. IEEE, 2010.

[53] Peter Leven and Seth Hutchinson. A framework for real-time path planning in changing environments. *The International Journal of Robotics Research*, 21(12):999–1030, 2002.

[54] Frank Dittrich, Stephan Puls, and Heinz Wörn. A Modular Cognitive System for Safe Human Robot Collaboration: Design, Implementation and Evaluation. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), WS Robotic Assistance Technologies in Industrial Settings*, page 10, 2013.

[55] Lucian Balan and Gary M Bone. Real-time 3d collision avoidance method for safe human and robot coexistence. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference On*, pages 276–282. IEEE, 2006.

[56] Ahmad Yasser Afaghani and Yasumichi Aiyama. On-line collision avoidance between two robot manipulators using collision map and simple escaping method. In *Proceedings of the 2013 IEEE/SICE International Symposium on System Integration, SII 2013, Kobe, Japan, December 15-17, 2013*, pages 105–110, 2013.

[57] Ahmad Yasser Afaghani and Yasumichi Aiyama. Advanced-collision-map-based on-line collision and deadlock avoidance between two robot manipulators with ptp commands. In *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, pages 1244–1251. IEEE, 2014.

[58] Ahmad Yasser Afaghani and Yasumichi Aiyama. On-line collision detection of n-robot industrial manipulators using advanced collision map. In *Advanced Robotics (ICAR), 2015 International Conference on*, pages 422–427. IEEE, 2015.

[59] Jianing Zhou, Kazuyuki Nagase, Shinya Kimura, and Yasumichi Aiyama. Collision avoidance of two manipulators using rt-middleware. In *System Integration (SII), 2011 IEEE/SICE International Symposium on*, pages 1031–1036. IEEE, 2011.

[60] J Zhu. *Real-time Collision Avoidance for Robot Manipulators by Braking Trajectory Prediction*. PhD thesis, TU Delft, Delft University of Technology, 2015.

[61] Fumi Seto, Kazuhiro Kosuge, and Yasuhisa Hirata. Self-collision avoidance motion control for human robot cooperation system using robe.

[62] Fan Ouyang and Tie Zhang. Virtual velocity vector-based offline collision-free path planning of industrial robotic manipulator. 2015.

[63] Cheol Chang, Myung Jin Chung, and Bum Hee Lee. Collision avoidance of two general robot manipulators by minimum delay time. *Systems, Man and Cybernetics, IEEE Transactions on*, 24(3):517–522, 1994.

[64] Su Il Choi and Byung Kook Kim. Obstacle avoidance control for redundant manipulators using collidability measure. *Robotica*, 18(02):143–151, 2000.

[65] Farshid Shadpey. *Force control and collision avoidance strategies for kinematically redundant manipulators*. PhD thesis, Concordia University, 1997.

[66] Cheng Fang, Alessio Rocchi, Enrico Mingo Hoffman, Nikos G. Tsagarakis, and Darwin G. Caldwell. Efficient self-collision avoidance based on focus of interest for humanoid robots. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 1060–1066, Nov 2015.

[67] R. V. Patel and F. Shadpey. *Control of Redundant Robot Manipulators: Theory and Experiments*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[68] Philipp Ennen, Daniel Ewert, Daniel Schilberg, and Sabina Jeschke. Efficient collision avoidance for industrial manipulators with overlapping workspaces. *Procedia CIRP*, 20:62–66, 2014.

[69] Dirk Biermann, Raffael Joliet, and Thomas Michelitsch. Fast distance computation between cylinders for the design of mold temperature control systems. 2008.

[70] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[71] Magnus Christian Bjerkeng. Coordinated control with obstacle avoidance for robot manipulators. 2010.

[72] Eckhard Freund and J Rossman. The basic ideas of a proven dynamic collision avoidance approach for multi-robot manipulator systems. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1173–1177. IEEE, 2003.

[73] M Sc Sami Haddadin. Towards safe robots: Approaching asimovs 1st law. 2011.

[74] Oussama Khatib. Force-based motion control of robot manipulators. In *Proceedings International Meeting on Advances in Robotic Kinematics, Ljubljana, Yugoslavia*, pages 176–180, 1988.

[75] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17(1-19):16, 2004.

[76] Ignacy Dulęba and Michał Opałka. A comparison of jacobian-based methods of inverse kinematics for serial robot manipulators. *International Journal of Applied Mathematics and Computer Science*, 23(2):373–382, 2013.

[77] Gilbert Strang. Introduction to Linear Algebra. *Mathematics of Computation*, 18(1):510, 1980.

[78] Shuzhi Sam Ge and Yan Juan Cui. New potential functions for mobile robot path planning. *IEEE Transactions on robotics and automation*, 16(5):615–620, 2000.