Realtime Motion Planning for Manipulator Robots under Dynamic Environments: An Optimal Control Approach.

by

Olabanjo Ogunlowore

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2013

Declaration of Authorship

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.
I understand that my thesis may be made electronically available to the public.

Abstract

This report presents optimal control methods integrated with hierarchical control framework to realize real-time collision-free optimal trajectories for motion control in kinematic chain manipulator (KCM) robot systems under dynamic environments. Recently, they have been increasingly used in applications where manipulators are required to interact with random objects and humans. As a result, more complex trajectory planning schemes are required. The main objective of this research is to develop new motion control strategies that can enable such robots to operate efficiently and optimally in such unknown and dynamic environments. Two direct optimal control methods: *The direct collocation method* and *discrete mechanics for optimal control* methods are investigated for solving the related constrained optimal control problem and the results are compared. Using the receding horizon control structure, open-loop sub-optimal trajectories are generated as real-time input to the controller as opposed to the predefined trajectory over the entire time duration. This, in essence, captures the dynamic nature of the obstacles. The closed-loop position controller is then engaged to span the robot end-effector along this desired optimal path by computing appropriate torque commands for the joint actuators. Employing a two-degree of freedom technique, collision-free trajectories and robot environment information are transmitted in real-time by the aid of a bidirectional connectionless datagram transfer. A hierarchical network control platform is designed to condition triggering of precedent activities between a dedicated machine computing the optimal trajectory and the real-time computer running a low-level controller. Experimental results on a 2-link planar robot are presented to validate the main ideas. Real-time implementation of collision-free workspace trajectory control is achieved for cases where obstacles are arbitrarily changing in the robot workspace.

Acknowledgements

Special thanks to those who were instrumental to the success of my work; my supervisor, **Prof** Soo Jeon for guidance and mentoring, **Prof** Victor M. Becerra for his audience and help with using his software. Positive influence Omar, Leaundra, Franny and my research group members: Kamal, Vahid, and Hyunki.

Dedication: To Christ Jesus, Mum and Dad.

Contents

List of Figures

List of Tables

List of Abbreviations

SQP	**Sequential Quadratic Programming**
ADOL-C	**Automatic Differentiation by OverLoading in C++**
PMP	**Pontryagin Maximum Principle**
LP	**linear Programming**
NLP	**Nonlinear Programming**
OCP	**Optimal Control Problem**
KOCM	**Kinematic Open Chain Manipulator**
DCM	**Direct Collocation Method**
LOCP	**Lagrangian Optimal Control Problem**
KKT	**Karush-Kuhn-Tucker**
RHC	**Receding Horizon Control**
DHOCP	**Discrete Optimal Control Problem**
DMOC	**Discrete Mechanics and Optimal Control**
UDP	**User Datagram Protocol**
TCP	**Transmission Control Protocol**
IPOPT	**Interior Point Optimizer**
AMPL	**A Mathematical Programming Language**
PSOPT	**Pseudospectral Optimization**
QOC	**Quadratic Optimal Control**
LAN	**Local Area Network**

Chapter 1

INTRODUCTION

1.1 The Kinematic Chain Manipulator(KCM) and Modern Applications

KCM's are robots with serially connected interacting link mechanisms where actuation is engaged accordingly at points called joints and the resulting motion is obtained by composition of the independent motions of each link with respect to the previous one. They make up over 50% of the robots in existence and are very common in automated industrial applications [2].

The motivation for considering this problem setup, *motion planning and dynamic obstacle avoidance in real-time*, is based on the prevailing existence of this class of robots in many modern engineering applications especially in many semi/fully automated processes.

Possible applications of efficient and effective motion planning of robots would be instances where output may be enhanced by robotic activities in dynamic environments such as human-robot, multi-robot, and objects-robots collaborative tasks. Humans working jointly with robots in a safe work envelope in terms of collision-free motion is an enticing venture and the flexibility of such applications has great economic implications particularly in highly structured industrial environments.

Example applications are in automobile assembling plants with human-robot interaction and multifingered self-guided surgery robots in medical application platforms where

properties such as precision, accuracy, repeatability, and speed are required.

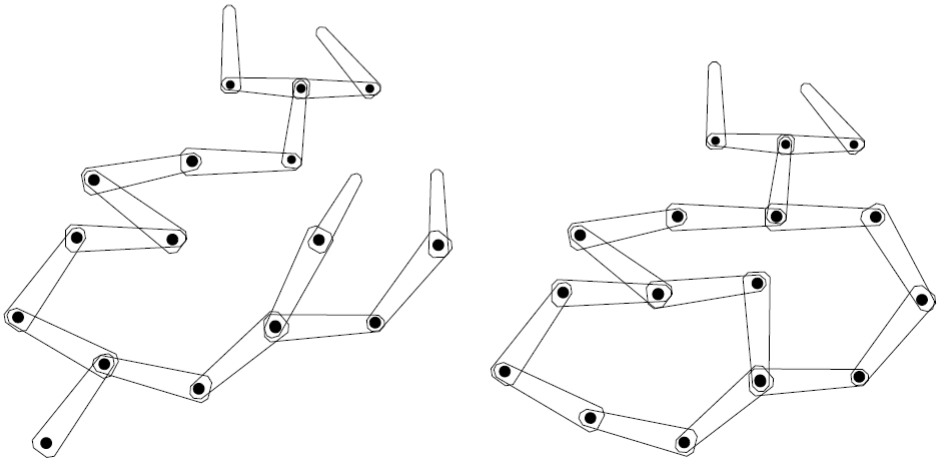

FIGURE 1.1: Serially connected chain of rigid bodies. Left: A fully open KCM structure. Right: A KCM with internal loops [3]

Usually the complexity in controlling this type of this robot configuration is based on its kinematic configuration i.e. the number of interacting links, which in turn dictates the systems dynamics i.e. the motion properties of the robot system and the mechanism enabling factors at the joints. Further constraints that may add to the complexity in achieving efficient motion control are obstacle avoidance (maneuverability capabilities) which is important for KCM's in collision-free application such as in[4]. This is the primary focus of this work.

The typical task of this class of robots is to move the end-effector of the manipulator from some initial point to a desired point in the most efficient way within its workspace. Efficiency here could include some or all of the following criteria:

- Motion applications for Zero-error tracking of a predefined path commanding a manipulators end-effector position.
- Motion control in shortest time interval and/or distance of a desired path or from one point to another.
- Maneuverability property around obstacles and fulfilling collision-free kinematic properties.

- Motion planning with least residual vibration.
- Minimum actuation (Torque command) during motion. etc.

Complex motion planning has to be considered to achieve these objectives.

While classical control approaches are well established as a comfortable means of achieving control of mechanical systems, they are no longer the best application for systems with highly complex inherent dynamics. In modern industrial applications of engineering control, complex systems with high level subsystems interactions and nonlinearities are difficult to deal with using conventional PD/PID controllers and more complex algorithms such as filters and estimators are augmented to capture the more complex systems dynamics and improve the system performance for .
It is difficult to design controllers for such system using the conventional classical controller models from traditional mechanical control schemes alone. This is because of the high nonlinearities in the systems dynamics which will usually result in loop coupling effects. For more complex scenarios especially in cases where obstacles have to be avoided, the above highlighted classical control scheme usually becomes insufficient and many-a-times needs to be greatly modified.
The need arises for more sophisticated controller and the intellectual zeal of continuously optimizing processes fuels this initiative. To produce realistic results, it may further evolve into incorporating controllers with manifold attributes. There is no generic approach to achieve this.

Optimization based approaches for control of constrained mechanical systems have been quite successful in the past, and as the complexity and the functionality of mechanical system becomes more stringent, these approach seems to be more adopted for higher performance in such applications as these.
This study dwells on optimized motion planning schemes for generating collision-free trajectories in real-time control applications for KCM's. It essentially dovetails into a twofold control problem. Optimal collision-free trajectory generation on one hand, and real-time control of the robot on the other. This is popularly referred to as on-line motion planning.

On the trajectory generation part, the idea is to adopt a class of direct optimal control methods: The Direct Collocation method (DCM) and Discrete Mechanics for Optimal

Control (DMOC) methods as the basis of transcribing a continuous optimal control problem into a corresponding discrete optimization problem and solved for local root solutions using a NLP optimization algorithm. Reducing the computation time of solution (optimal controls and trajectory) is paramount in this work because the solutions must be adoptable for practical real-time control applications.

1.2 The Proposed Structure for Achieving Real-time Collision-free Control:

- Establish bidirectional communication between two dedicated computer processors. Using a two-degree of freedom control structure, input commands for a real-time servo controller in form of optimal collision-free trajectories computed by an optimal control solver are implemented for motion spanning of a test-bed experimental robot in a feed-forward manner. The servo-controller hence spans the robot along the desired collision-free path.
- A hierarchical network control ensures proper activity ordering of the synchronized sub-systems. The overall setup allows for the possible exploitation of better processor speeds of the independent sub-systems hence enhancing their performance for the intended real-time motion coordination.
- The hierarchical network control structure based on some case structures and rules, sequences the activities of this interacting sub-systems processes to ensure the integrity of the solutions as feasible, optimal, and real-time during the robot runtime.

The goal defined as an optimal control problem is to find a set of control actions that fulfills the minimization of a chosen performance index subject to the dynamics of the system and other constraints (path, geometric, boundary conditions) which gives the most efficient trajectory along a specified time varying path or point-to-point motion.

To validate the process experimentally, a low-level real-time computer is designed as a torque computing controller for the robot joints actuation using the configuration state variables of the optimal trajectory generated as input commands. For this study, the

links flexibility effects are cleverly ignored in the problem model. This is because it complicates the dynamics of the systems substantially and is not a related addition in the model problems studied. The model employed here is still sufficient to study and investigate the methods adopted in this work.

1.3 Avoiding Dynamic Obstacles

Maneuverability around obstacles with time varying properties is the main focus of this study. The solution for this problem would be much easier if the information of all possible future environment variables with respect to obstacle location in the robot workspace are known prior. This assumption is far from realistic in real world applications as the location of potentially changing obstacle(s) cannot be forecast with reasonable precision. This brings up the issue of offline versus online motion planning for trajectory generation with respect to time varying obstacle coordinate locations.
With real-time motion control, one may confidently adopt a better planning method which fulfills scenarios with arbitrary moving obstacles as well as the non-violated kinematic and dynamic constraint on the system, unlike offline motion planning which is static and cannot capture arbitrary and dynamic scenarios.

With the new age of faster computer processors, modern optimization algorithms with consistent convergence properties and well-established control structures, such real-time control applications may be realized if speedy computation of feasible solutions can be accomplished. To accommodate this time-varying obstacle instance, a time based approach is adopted. The receding horizon control framework is used to compute sub-optimal collision-free trajectories in time fractions, so as to capture the dynamic property of the obstacles and progressively iterate the process to achieve desired motion perpetually.
The primary constraint is the dynamics of the system specifying its motion properties and kinematic constraints, which specifies the geometric design of the system. A difficulty in coming up with the solutions of these sort of problems quickly is the nature of the constrain sets with respect to its degree of nonlinearity in the systems dynamics and the size of the constraint. Other constraints in this case are the obstacle path constraints which have to be avoided and the bounds on the state and control based on the

parameters of the robot geometry and actuator structure.
The reasons for this difficulty are well explained in chapter 3 where the problem is fully modeled and is formulated mathematically. The optimal control techniques adopted are motivated most especially because of the nonlinearities in the systems dynamics and how they may be advantageous and exploited in computing trajectories fast and accurately enough for real-time application.

1.4 Outline

Here in chapter 1, a general introduction and the motivation to the problem is expressed with a brief summary of the proposed solution approach.
Literature reviews on past and existing state-of-the-art methods in motion planning algorithms, optimal control methods and optimization schemes related to motion planning for mechanical systems similar to the KCM application are shown in chapter 2.
Chapter 3 highlights the detailed explanation of the methods employed for achieving the motion planning task for the N-link KCM robot. The receding horizon optimal control problem is stated.
Chapter 4 illustrates the methods and solution structure of a 2-link KCM. A redundant linked property KCM is used to generalize the problem formulation for N-linked manipulators. Simulation and experimental results follow in chapters 5 with comparison of the optimal control methods used.
Observations, final conclusions and future postulated directions for further work make up the closing of this thesis in Chapter 6.

Chapter 2

LITERATURE REVIEW

"As long as the paradigm of qualifying controls as better or worse is a foundation of Engineering thinking, optimization will serve as the vehicle for conveying ideas into solutions"

Henryk Grecki

This chapter examines past methods which have been applied in the realization of motion planning in mechanical systems similar to the Kinematic Chain Manipulator robot. It especially focuses on path planning and trajectory generation algorithms, and methods in the optimal control domain and control structures for accomplishing real-time motion control.
All through this report, A path in terms of motion coordination may be defined as the state coordinates of a system in terms of its position only while a trajectory is the path followed by the system in addition, the time profile along that path.

Large number of control problems for mechanical systems are based on controlling the position or location of a mass using a force or torque as the input variables. Instead of the pure regulation problem of driving the output location to a specific value, the position of the mass is often required to follow a prescribed path [5] in light of reducing some related cost function and fulfilling some underlying physical constraints of a system. This is the general underlying concept of optimal control methods for motion spanning in mechanical systems.

Within the scope of optimal control for motion control in articulates manipulators, trajectory planning as a motion optimization scheme involves designing optimal motion commands to minimize some performance index subject to the dynamics and geometric constraints of the system. To make it realistic, some prescribed constraint limits may be imposed on the state and control variables of the system.

The documented study of articulated manipulator control goes back in the mid-seventies [6, 7]. Although the system attributes are getting more complex by the day, the goal has mostly been the same; deriving optimal and efficient control laws to achieve timely and precise coordination schemes dictating the robots motion.
Various endeavors have been employed to achieve this and are discussed as follows:

2.1 Path Planning Algorithms for Articulated Manipulators

Path planning algorithms have been existent for many decades and have been successfully applied in motion coordination schemes in mechanical systems. They are limited to motion coordination in terms of position only and a framework is shown below. They are usually classified as knowledge-based driven planning algorithms.

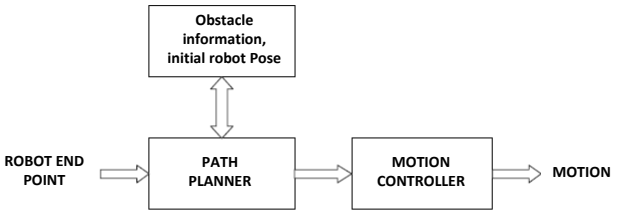

FIGURE 2.1: Framework of robot motion planning using Path planning knowledge based algorithms

Three of these methods which have been particularly adopted in recent times for use as motion planning schemes in mechanical systems similar to KCM application in dynamics environments are: The **Probabilistic Road Map(PRM)**, the **Rapidly Exploring Random Tree (RERT)** and the **Elastic Strip Methods (ESM)**.

The ***Probabilistic Road Map*** is a type of randomized sampling technique which generates non-violating maps based on predefined forbidden regions. The entire work envelope of a robot configuration (joint or workspace) is spanned to derive collision-free local roadmaps from an initial position to a desired point.
It goes through a learning phase where a roadmap is made from nodes denoting interacting points of an object versus its surrounding environment represented as edges. From an initial configuration of the robot manipulator, a test for collision is done and a near configuration is picked randomly afterwards. The collision-free test is done repeatedly at successive iterations using a newly generated roadmap design as the nodes and edges are connected to span the robot to a final point resulting in collision-free paths. Applications to the robot manipulator can be found in [8, 9].
This planning scheme is quite efficient as a local planning scheme, however when real-time motion is considered, it is unsuitable because it suffers a great increase in computation time due to the algorithm searching the entire problem domain.

The ***Rapidly Exploring Random Tree*** method is based on a randomized tree-search step which constructs branches towards unexplored regions in a space. At each iteration of this process the tree is extended by randomly selecting states and adding branches accordingly. Three possible scenarios are arrived at:
Reached, meaning the branch has successfully reached a destination state.
Advanced, meaning the state in question is not the destination/final state and further branching is employed.
Trapped where a state is infeasible based on some predefined rules.

Although the randomness in the choice of tree branching helps reduce this computation time, this method when real-time motion control is considered is still not desirable enough as it still takes a long time to explore all the branches.
Hybrids of this method can be seen in [10] where a two-step setup is designed with the popular Support Vector Machine(SVM) structure for directing the tree-branches in the best direction reducing computation time. Similarly in [11] key configurations are used to guide tree branching process by connecting the start configuration to the final desired pose of the KCM.
In [12, 13], a bidirectional RERT is used rather than a randomized selection which proves to help make the solution converge faster with simultaneous trees been built in opposite

directions and rules used to discharge vulnerable trees.
Also, a well applicable variant of this method applied in real-time control applications of robot manipulators is stated in[14] where arbitrary moving obstacles can be considered during runtime. The drawback in the successful application of this work is that the generated paths cannot be proven to be optimal.

Oliver et al in [15, 16] proposed a motion planning algorithm specified for robot obstacle avoidance/maneuverability procedures known as the ***Elastic strip method***. Obstacles in a robot environment are represented as a one-dimensional curve in the robot joint or workspace and connecting collision-free smooth paths are generated. The elastic property of the path allows for it to deform by reacting accordingly to the obstacle coordinates changes and hence preserving a collision-free path. This method showed success for manipulators with many degrees of freedom. This planning scheme still fall short as qualifying candidate methods for generating collision-free paths with globally optimal properties.

The three methods highlighted above can be generally classified as motion planning algorithms and their variants as reactive motion execution algorithms which use some rules or potential field-based measures to modify the base algorithm to reactively avoid forbidden regions. They all have been used in various cases to derive feasible trajectories for manipulators, but all suffer the issue of no explicit strictness of these paths being optimal ones. They are all local path planning algorithms and when real-time objectives are in question, they fall short due to their lengthy solution time computation.

2.2 Trajectory Generation for Nonlinear Mechanical Systems (Optimization techniques)

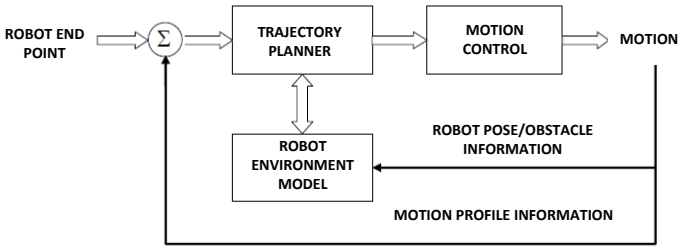

FIGURE 2.2: Framework of robot motion planning using trajectory planning schemes

Computationally efficient trajectory optimization is an enabling technology for many new facets of engineering [17]. This made research in this field lucrative and has received its due attention in the past decade.

A well applied method of optimization in trajectory generation problems is the *Interior point method* which was popularly accepted after a lot of re-modification and applied to cases in linear programming (LP) problems. It was originally introduced in the Karmarkar's algorithm [18] and with more development on his works in later years it proved to solve convex problems in considerably small polynomial time with respect to other methods available at that time.

This claim is justified in further works by this founding author in his 1991 reports [19] which showcased the exploitation of sparsity property of Jacobian matrices for computing solutions of LP problems faster. The success of this method was most recognized with convex type LP problems and successful application to NLP problems was still far-fetched.

In terms of real-time control, this algorithm is quite popular for optimal control problems in aircraft control applications. The successes with respect to this algorithm for various optimal control problems are tangible; however, another one of its drawback is in the aspect of linearizing the systems dynamics. This complexity of the linearization process increases the amount of information loss of the system which is detrimental to

the quality of derived root solutions of the resulting LP.

The major difficulty in the application of this method for the articulated manipulator robot as shown in [1, 20] lies primarily in the complexity of the nonlinear equation describing the system dynamics. In addition, imposition of various forms of state and control variable constraints such as torque limits (controls) and path constraints specifying forbidden regions explodes the size of the problem greatly and makes its solution more difficult to compute.

This makes this approach unattractive for such a problem and in summary, the results achieved from primal dual Interior-point algorithm are more suitable for convex rather than NLP problems.

Recent modification to this base algorithm and works in relation to nonlinear programming in recent times can be appreciated in the works of Anthony V, P. McCormick Et al in [21] where the algorithm was extended for use in Nonlinear Programming problems. It is the basis of the widely successful Interior Point Optimization (IPOPT) solver which is a Sequential Quadratic Programming (SQP) algorithm and extensively used in this work as the optimization algorithm for finding local minimums of the resulting NLP's.

Researchers such as Wassem A et al, Arthur R and Jonathan P [22] and Cedrics MA et al in [23, 24] have used the ***Mixed Integer Non-Linear Programming*** (MINLP) strategy for optimal path planning and trajectory generation for various applications related to KCM's in recent times. The popularity of the MINLP algorithm in the application of optimal path planning is the suited formulation of the method for problems with imposition of considerable multiple constraints for obstacle avoidance. A major idea of the MINLP approach in modeling an optimization problem lies in the handling of non-convexity.

Where a linear method may not guarantee any form of optimal solutions, the MINLP guarantees some globally optimal one when handling constraints for disjointed sets [24]. They can be easily handled and represented using MINLP as binary or mixed integers. The method is also particularly well suited for this kind of control problem because the MINLP structure can encode discrete decisions for the control and state variables and non-convexity in form of integer constraint. The algorithm has been proven to also perform better with extensions of some artificial intelligence algorithms but with the trade-off of a very sophisticated problem design. Coming up with effective heuristics for

non-convex MINLP is a difficult task and finding feasible solutions can be quite tricky. Hao .D, Gunther and Olaf Stursberg in their articles [25, 26] also used the MILP for coordinated motion planning in articulated robots for obstacle avoidance. It showed a large size of binary constraints been generated and endeavors were taken to reduce this by some geometric representation method with respect to the shape of the obstacle. The results were still inconclusive for a case where the shape was not important and the computation time suited only an offline solution where environment information was known only prior to generating the optimal paths.

Hierarchical Trajectory generation has also been studied as a class of motion planning methods for mechanical systems with nonlinear systems dynamics [27]. The major drawback with this method is the unrealistic assumptions with respect to various constraint handling forms. The level of complexity of the nonlinear system greatly affects the outcome and only with small and abstracted system is this method guaranteed to be feasibly applicable.

These optimization based methods used for trajectory generation have been very successful in offline computation of solutions. The results are quite attractive when offline solutions may be sufficient but not so for real-time control and especially with time varying variables because of the assumptions in projecting the trajectory planning problem onto a related control system of smaller dimension.

The results can be implemented in only zero intelligence scenarios such as static environments and not in an arbitrary changing environment. The method only proved profitable for cases where the full information about the system must be known prior to the start of the planning process, which is usually not the case [28].

On a more realistic note, in [29] real-time trajectory generation and tracking of nonlinear control system with a two degree of freedom technique was studied. A nonlinear tracking problem which takes care of the trajectory generation on one hand, and on the other, a local gain scheduling scheme for position control. This problem setup had no constraints (systems dynamics) so called non minimum-phase problem differing from that employed in this thesis which is a phased problem and with the systems dynamics considered as constraint.

Another method is the ***External Active set Strategy*** [30]. This employed the use of outer approximations of constraint sets by ignoring constraints which were redundant or

were isolated based on the systems kinematic geometry. This resulted in a considerable reduction in total active constraints generated and hence less computational time. It is most suitable for a class of problems with a large number of inactive inequality constraints sets.
It was introduced in [1] to reduce the constraint set by neglecting those constraints which may never be reached due to interaction of the kinematic structure of the robot with respect to the set of obstacles at a specific time instance.
Results were consistent and accurate but the issue was still the lengthy computation time. It proved successful in this regard, but not so well-off for real-time control scheme as solutions were derived in none less than ten seconds.
The technique is however comfortably applicable for offline trajectory generation problems.

Learning based control using Neural-Networks have also been used to achieve real-time dynamic control of manipulators using the complexity of the robot internal kinematic structure as trajectory performance learning. Bulky computation time is the major drawback in this adaptive controller scheme as performance is state variable dependent and the computation time grows at an overwhelming ratio to the system state variables [31].

This above outlined issues however created a window for the conceptual methods to be created and modified into forms applicable for typical control problems to be solved using numerical methods hence it increased the interest of engineers in adopting this well founded optimization methods and in recent times, have fostered the implementation of solutions even in real-time control applications to a comfortable level.

2.3 Optimal Control Approaches for Trajectory Planning

In recent times (about a decade), various hybrid optimal control techniques have been introduced to tackle the problem of articulated manipulator robot control from various angles. Hybrid here could be referred to as techniques with multiple or integrated concepts in numerical computation and optimization.
Some results from techniques in the optimization dimension applied over the years to mechanical systems which have close adaptation to KCM's are briefly explored below:

One of the earliest endeavors of optimal control schemes in association with robotics and in particular articulated manipulators was around the nineteen-seventies where an optimal control algorithm was developed to solve Quadratic Optimal Control (QOC) problems. The dynamics of the system was a lone constraint with no other constraints on the state and control variables. Computation of explicit solutions for the system was accomplished [32].

The simplicity of the problem rendered the solution quite impractical for real life applications as that ventured in this report where robots relate with dynamics environments and therefore have a lot more constraints to fulfill. This however gave way to the possibility of obtaining solutions of optimal control problems using numerical methods in KCM's.

A similar study in [33] showed the performance of an algorithm based solution for QOC problems in a 2-link flexible manipulator for a geometrically constrained case. The solution showed better results in comparison to PD/PID control schemes showing classical control methods were inferior, moreover the implementation was also practically viable.

Another approach for trajectory planning of mechanical systems is the H^{∞} method [34]. Using 3rd order spline curves, the initial and desired final configuration of the system are constructed to derive collision-free feasible trajectories. The H^{∞} optimal control design was used to compute state feedback control laws deriving optimal trajectories from the set of earlier generated feasible trajectories.

This class of methods for more interested readers may be found in [35–37]. Unmanned Aerial Vehicles (UAV) control and trajectory planning for static and dynamic environments is a well-established area of control where these above highlighted optimization methods and conceptual schemes are used for computing and applying solutions in real-time. The formulation of the systems dynamics of UAV's and articulated manipulators are similar and the methods developed in this field look promising for robot manipulator control.

On a general note, when solutions are sufficiently not time dependent, purely optimization methods are very attractive as they may give very accurate results. An important part of this work as is many similar works, is to examine solutions from various standard optimization algorithms and restructure the problem and proposed solution algorithm

for possible reduction in computational time.
In achieving this, the endeavor here will be the representation of the problem by exploiting its known properties; both mechanical and geometric ones.
Hartl et al[38] states the common perception of engineers to optimization: "Optimal control problems with state variable inequality constraints are not easy to solve, and although the theory is not ambiguous, since there exist various forms of the necessary and sufficient optimality conditions because the literature on this subject is not comprehensive, at times, incorrect or incomplete, it has been hard to understand, especially for people working in applied areas".

The techniques for solving optimal control problems can be classified into two methods:

1. Indirect (analytical) methods: When solutions to optimal control problems are found directly by deriving the fulfilling conditions from the calculus of variation (Pontragin's Maximum Principle), such a procedure is termed an analytical method.

2. The other form of finding solutions to OCP are direct methods. The optimality conditions are equivalent to the case of the indirect methods however for this class of methods, the idea is to convert the original infinite dimensional continuous optimal control problem into a finite one with state bounds by time discretization of the state and/or control variables making up the partial or ordinary differential equation and integral terms of the systems making up the systems dynamics constraint and objective function respectively.

2.3.1 Indirect Approach to Solving Optimal Control Problems

Optimal control problems were initially solved by using the indirect method. This is done by forming a set of necessary conditions and then solving these equations usually partial or ordinary differential equations. There are numerous processes to achieve this. Pontryagin's Maximum Principle based on the calculus of variation is one. These optimality conditions are usually formulated as initial or boundary value problems represented by differential algebraic equations.
These solutions are usually approximates at the best and take a considerable amount

of time to compute depending especially on the complexity of systems dynamics. The following works [39–41] gives a class of OCP's and solutions using the indirect approach.

In [42, 43], the authors derived solutions for Lagrange systems using parameter projection algorithms and Hamilton-Jacobi (H-J) equations with inequality constraints, for solving a class of nonlinear problems having parametric uncertainties and external disturbances. The work proposes a design method for an adaptive, reliable and robust control by explicitly solving these H-J equations. The results showed the possibility of using the concepts of theoretical nonlinear programming as optimal control method for holonomic systems but the drawback was the high complexity in the problem formulation. Two coupled nonlinear Ricatti equations had to be solved to form a closed form solution which is very inefficient for practical applications.

This is how far closed-form and indirect methods will be discussed as the prospect seems to lie in direct approaches if real-time control is the aim.

2.3.2 Direct Approach to Solving Optimal Control Problems

Generally, procedures which convert OCP's into Nonlinear Programming (NLP) problems are regarded as direct methods through time discretization of chosen optimization variables and so, a discrete time optimal control problem is typically a NLP problem. In more recent times, the approach used to solve and implement optimal control problems has drifted towards direct approaches. Numerous reasons account for this.
One is the advancement in the computational power due to advanced technologies in microprocessor and the development of a great number of efficient numerical computation schemes and hybrid optimization algorithms (integrated optimization techniques enhancing solution features) which suits many classes of optimal control problems. Another reason for its general acknowledgment is that the mathematics of converting optimal control problems into simple optimization ones is also very well understood and has become standardized over the years.
This procedure is also not always a first choice technique for solving optimal control problems for some drawbacks. A popular one mentioned among control engineers and unequivocally transmitted in [44] is the high level of mathematical sophistication usually explored to formulate complex engineering problems into an optimization one, the doubtful viability of optimization under certain conditions, measurement problems, and

sensitivity of optimal solutions to minute variances such as disturbance and modeling errors.

With respect to the implementation of the solution, a major concern and dependence on the integrity of the solution especially in terms of computation time (especially for real-time/online optimization), is the hardware configuration which greatly affects the viability of the solution.

2.4 Candidate Direct Methods and Techniques for Online Trajectory Planning

Continuous time OCP's are more difficult to solve compared to discrete time OCP's primarily because the solution of the problem requires bound on elements of infinite dimension rather than finite dimensional vector spaces.

This difference introduces a new set of approaches to solve the problem by converting the continuous optimal control problem into a discrete one which is a NLP. A nonlinear programming algorithm such as a SQP can be used to find root solutions. This process is known as time discretization and is the underlying principle of direct methods.

A class of direct methods are the Shooting methods (Single and Multiple), Direct Collocation Method (DCM) and the Discrete Mechanics for Optimal Control (DMOC).

2.4.1 Shooting Methods

Shooting methods approximate a finite solution by time discretizing the control variables in the OCP of the mechanical system in question. In the case of the single shooting method, an initial guess is formulated from the vector set of the control variables, and the optimal states forming a solution trajectory is arrived at using the control parameters only as basis for formulating all optimization variables to be minimized.

The multiple shooting is similar, but with the added reordering of the solution at subintervals of time (discrete points) along the minimization path. At each discrete point, additional optimization variables are included to meet some sub-boundary conditions in the form of constraints (inequality and equality) with respect to the state to ensure a continuous flow along the optimization path.

2.4.2 Direct Collocation Method (DCM)

In the DCM method, the nature of the high nonlinearities in the dynamic equations governing the systems motion is exploited. The nature of the resulting sparsity property of the higher order derivatives (Jacobian) in such systems with highly nonlinear dynamics is noted and this motivates the discretion to adopt the Direct Collocation Method formulated by Hargraves and Paris [45] as a discretization setup.

Linearization is not desirable when the dynamics of a system is governed with high nonlinearities. Discretization seems to be a better approach as it preserves the dynamic properties of the system unlike linear approximations.

In the ***Direct Collocation*** approach; the optimal control problem is formulated by a time discretization of the system parameters in terms of both the controls and state variables. The problem is solved by approximating the ODE's describing the OCP using piecewise defined higher order polynomials and solving the problem at discrete collocation points called nodes along the path to be minimized.

Suitable numerical quadrature rules approximate the integral function hence replacing it by close approximates as the dynamics are collocated at specific set points in the time interval of a finite sub-space. This is pretty much an established and a foolproof theory for discretization and it is the rationale for the popular mesh based theory in finite element analysis [46].

This also results in an NLP like the shooting methods, but in this case simultaneously exploring both the state and control variables as optimization parameters gives additional inequality constraints at the nodes due to the discretization process.

The major attraction of the *Direct Collocation* method is that it handles problems with state and control bounds and the high ordered capturing of very nonlinear systems dynamics (ODE's) using high order Legendre polynomials has been proven to be very effective [47–49]. The solution can also be potentially derived at a much faster rate as the ODE simulation and the optimization parameters are solved simultaneously compared to the Shooting methods.

The approach is that the systems dynamics is parameterized and the optimal control problem is solved as a nonlinear programming problem. Normally, this transcription process is used to convert a continuous optimal control problem through discretization into a nonlinear programming one. This results in a much larger problem size as the

equality constraints are now functions of both the state and control variables and could be an inefficient representation of the optimal control problem [34].

The benefit however is that the high nonlinearity of the systems dynamics creates a high degree of sparsity in the constraint Jacobian and the solutions of the resulting NLP, therefore solutions may be reached at much faster computation time [50].

The accuracy of the solution is then dependent on how well the higher-order polynomial functions can capture and approximate the entire function between successive collocation points of the finite discretized problem setup [51]. A good estimation of the costate variables using some well-established quadrature rules also makes it a good choice as it ensures its immunity to ill conditioning in this regard.

2.4.3 Discrete Mechanics and Optimal Control (DMOC)

The ***Discrete mechanics for Optimal Control*** approach fully presented in the work of J.E Marsden and M.West [52, 53] is quite similar to the collocation method in terms of the discretization and the resulting NLP which is derived but with a smaller problem size. Unlike the collocation method where the cost function and the equation of motion representing the dynamics of the system are discretized, the variation of the action sums of the Lagrange mechanics using the system geometric arguments and the cost function are discretized directly using carefully picked variational time integrators right in the systems configuration space [54].

It uses variational integrators to form time-stepping schemes which in turn preserve the geometric properties of the system during discretization. The governing principle describing the system is discretized directly. Similar endeavors to that of this work and general applications for mechanical systems are found in [55–57].

This method (DMOC) is also attractive because of the structure of its formulation. It redefines both the states and control variables of the system as the system parameters and optimization variables in a simple straight forward manner with the aid of time-integrators with configuration space variables only.

2.4.4 Sequential Quadratic Programming (SQP)

All the above direct methods essentially transcribe an original continuous infinite time Optimal Control problem into a finite dimensional NLP. At this point a local root finding algorithm can be used to find the local minimum to this NLP problem. One of such is the SQP algorithms.

Largely, NLP problems can be solved by finding the solution to a sequence of quadratic relationships of a system differential equation using a method like the SQP.

The process is achieved by formulating an NLP at given approximate solutions by a quadratic programming sub-problem; a Lagrangian of the Non-linear program subject to the systems constraints (linearized). Good convergence properties have been established by this method in the past. The SQP uses a merit function to formulate in progressive steps towards finding a global minimum [58–60]. This is iteratively done to create a sequence of approximate solutions that leads to the final local minimum. The SQP methods have been implemented in many packages, including IPOPT®, NPSOL® and MATLAB®. IPOPT® is the solver used in this work because of the good inter-phasing compatibly with the sub-systems (hardware, programming language and operating system) used for the problem setup. It is also a generally applauded as a very efficient SQP algorithm especially for problems similar to that in this report.

2.4.5 Receding Horizon Control (RHC)

In robotic manipulator control, using conceptual applications schemes such as the RHC for implementing control processes is not common knowledge and is investigated in this work as a technique to capture the dynamic changes in the robots environment.

The idea of batch solutions became popular as a means to achieve fractional or part solution updates for systems and the evolution of this scheme brought about some successful works. Good formulation of this conceptual method which help implement optimization processes for many classes of control systems are well documented in [61, 62]. The idea is to compute solutions of a problem model within a suitable time frame capturing all relevant system variables (prediction horizon) and implement its solutions on a fraction of the prediction horizon as input commands for the model plant. The problem is solved iteratively until the prediction horizon is exhausted as it moves in a receding manner towards the terminal state of the problem and the control commands computed for the

entire problem in sub-intervals of time.

The solution to the RHC problem provides a structure for suitably dictating the appropriate set of control inputs to be applied over a forecast or prediction horizon based on a current sampling interval and environment variables to sustain a safe control period over a fraction of a prediction horizon. This also means the performance of the RHC for a real-time application is wholly dependent on the pace the Nonlinear programming algorithm comes up with solutions to the problem in question.

2.4.6 Hierarchical Network Control

A common control structure to achieve real-time control in robot systems of this nature is to have both a optimal trajectory algorithm and a low-level controller design on a single platform. This structure has been used in various endeavors of real-time and online solution update. The issue is with the processing power of the microprocessor doing these two tasks simultaneously. It is usually not sufficient.

Hierarchical Network Control is a control structure dictating the precedence of actions to be taken where independent platforms performing specific actions need to interact or combined to achieve an overall single task. This application is popular in systems control where sub-system actions need to be optimized or enhanced to achieve better overall performance [63].

The method adopted in this work is synchronizing sub-systems through an adaptable network control system for instantaneous data transfer and communication between the independent platforms: The real-time processor running the robot controller gets input commands from the computer generating optimal trajectories and using hierarchical control, triggering action between the two sub-systems in terms of ordering precedence activities is controlled to achieve an overall real-time collision-free motion coordination for the robot in question.

The objective considered is to generate an optimal trajectory followed by a command solution update for the real-time controller. The interaction between the two subsystems needs a time critical switching medium for real-time synchronous activities to be possible. The implementation of the hierarchical Network platform and data transfer speed will hugely influence the performance of the overall system. This is achieved using a connectionless data transfer technology; the User Datagram Protocol (UDP) which continuously establishes reliable bidirectional subsystem interaction and transfers

information across the subsystems network in form of data packets. Timed hierarchical control triggering actions and rules specify the direction and data type (control command or robot environment variables) as the case may be.

Chapter 3

METHODOLOGIES

In this chapter, the methods for solving the collision-free trajectory generation problem for an N-link KCM in dynamic environments are explained. The optimal trajectory generation problem is solved using two direct optimal control methods. These two direct methods are the *Direct Collocation* and the *Discrete Mechanics* approached for solving OCP's method. Optimal control direct methods are characterized by a time discretization process which transcribes a continuous infinite dimensional optimal control problem into a discrete optimal control problem with finite dimension. The reformulated discrete time OCP equivalent to a Nonlinear Programming problem can in turn be solved for local optima's using root finding algorithms. A SQP based algorithm; The Interior Point Optimization (IPOPT) [58], an open source C++ based library is employed here. The theory of these direct methods and the steps for implementing them (DCM and DMOC) are discussed.

The receding horizon control framework is explored to condition the optimal control problem for sub-optimal computation in the light of capturing the time-varying properties of obstacle/path constraints. This ensures a platform for implementing a safe control horizon for a set of collision-free sub-optimal state/control at each phase of problem solution setup.

A cost functional representing the system parameters which is to be minimized, and a set of ODE's representing the dynamics, path constraints and the systems optimization variables are discretized to form a discrete time optimal control problem. The resulting solution to the resulting gives the desired collision-free optimal trajectory.

3.1 Problem Formulation: The N-Link KCM Problem

Considering the motion of an N-link KCM along a continuous desired end-effector trajectory where q_i specifies the systems coordinates (joint parameters) in vector form.
τ_i denoting forces in form of torque acting on the system at the links specified using the joint parameters l_{th}, specifies the links with respect to the base/fixed frame.
The trajectory planning task as an OCP falls under a minimum energy and an error tracking problem.
The goal is to move the system along a curve $q_i(t) \in Q$ within an initial and final time interval $[t_i, t_f]$ from an initial state $(q_{t_i}, \dot{q}_{t_i})$ to a final state $(q_{t_f}, \dot{q}_{t_f})$ in a closed region of the systems configuration space defined in: $Q \in \mathbb{R}^N$ where the Nth link of the manipulator is fully defined under the influencing forces $\tau_i(t)$ minimizing the energy sum of the entire systems $q_i = q_1, ..., q_N$ and the 'dot' notation to represent derivatives with respect to time.

3.1.1 The System Dynamics as a Constraint: Equations of Motion

The motion is constrained inherently by the dynamics of the manipulator, the kinematic constraint from the link geometric configuration and the time varying obstacle(s) coordinate locations in the workspace.

The general equation of motion of a Kinematic Chain Manipulator specifies the dynamics of the system and is the inherent constraint of the system which gives the response of the system to external forces. The dynamic equation i.e. the equation of motion of a manipulator may be derived by specifying the dynamics of the time evolution of the system subject to holonomic constraints.
There are quite a number of procedures to represent the dynamics of an N-link articulated manipulator, but for our problem (minimum energy and trajectory tracking for trajectory optimization implications), the Lagrange equation is suitable to formulate the total energy relationship of the system.
The end result is a formulation specifying the dynamic properties of the system in terms of the joint space variables; position, velocity, acceleration and its resulting forcing relationships.
An elegant way to construct and represent this especially for multi-link robots is using

the principle of conservation of energy. This is referred to as the Lagrange approach. Choice trajectories may then be generated at the task space level using the joint parameters and with forward kinematics computation it may be extended to workspace motion planning as desired.

The Lagrange equation of an N-link Manipulator is derived as follows:

Kinetic Energy of an N-link Robot: Taking the moment of inertia I_i about the center of mass of each link, with total mass m_i relative to a generalized coordinate frame $x, y, z \in \mathbb{R}^3$ attached to the center of mass, specifying the translational and rotational velocities derived with respect to the joint variable. From the appropriate Jacobian, the linear velocity of the links (J_{vi}) and the angular velocity of the links (J_{ω_i}) at the base where $q_i = \{q_1, ..., q_N\}^T$ is the vector at each joint angle, can be formulated as follows:

$$\begin{aligned} v_i &= J_{vi}(q)\dot{q} \quad \left[v_i \in \mathbb{R}^3\right] \\ \omega_i &= J_{\omega_i}(q)\dot{q} \quad \left[\omega_i \in \mathbb{R}^3\right] \end{aligned} \tag{3.1}$$

The generalized inertial tensor matrix in a 3-dimensional Cartesian coordinate system can be specified as:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \tag{3.2}$$

In state space form, total kinetic energy (T_T) can be represented as:

$$T_T = \frac{1}{2}\sum_{i=1}^{N}\left(m_i||v_i||^2 + {\omega_i}^T I_i \omega_i\right) \tag{3.3}$$

Potential Energy of an N-link Robot:The potential energy is essentially the gravity effect on the system. Taking each link inertia as in the case of the kinetic energy about the center of mass, the total potential energy V_T can be directly described as:

$$V_T = \sum_{i=1}^{N} m_i g l_{ci} \tag{3.4}$$

The Lagrange simply specifies the conservation law of motion which is the difference between the total kinetic energy and total potential energy acting on the system.

$$L(q_i, \dot{q}_i) = T_T(q_i, \dot{q}_i) - V_T(q_i) \tag{3.5}$$

The Euler-Lagrange (E-L) equation can then be easily derived from the Langrage equation which fully specifies the full motion properties of the system.

Substituting the equation of motion into the Lagrange equation and taking respective partial derivatives, the equation of motion of the system referred to as the Euler-Lagrange equations of motion is derived.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \left[\frac{d}{dt}\frac{\partial}{\partial \dot{q}_i} - \frac{\partial}{\partial q_i}\right](T_T(q_i, \dot{q}_i) - V_T(q_i)) = \tau_i \tag{3.6}$$

The Euler-Lagrange equation of the system of motion is given as:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \tau_i \quad i = 1, \ldots, N \tag{3.7}$$

3.1.2 Dynamic Equation: State Space form

State space equation denoting the systems dynamics can be written from the E-L equation as follows:

$$\tau_i(t) = M(q(t))\ddot{q}(t) + C(q(t)\dot{q}(t))\dot{q}(t) + g(q(t)) \tag{3.8}$$

The following parameters give more details about the variables involved in the dynamic model;
$M(q) \in {}^{N\text{x}N}$ Symmetric and Positive definite mass Inertia matrix.
$C(q, \dot{q}) \in {}^{N\text{x}N}$ Coriolis/Centrifugal forces matrix (Skew and Symmetric).
$g(q) \in {}^{N}$ Gravitational forces.
$\tau_d(t) \in {}^{N}$ Model disturbance
A nonlinear state space representation of the systems dynamics can be written as:

$$\dot{x}(t) = f_i(x_i(t), \tau_i(t)) \tag{3.9}$$

with

$$x\left(t\right) = \left[\begin{array}{c} q\left(t\right) \\ \dot{q}\left(t\right) \end{array}\right]$$

The robot position, velocity and acceleration are denoted respectively as $q\left(t\right)$, $\dot{q}\left(t\right)$ and $\ddot{q}\left(t\right)$ and driven by control forces at the joints $\tau\left(t\right) \in {}^{N}$ associated by the above dynamic equation.

$$\ddot{q}\left(t\right) = M^{-1}\left(q(t)\right)\left[\tau\left(t\right) - C\left(q\left(t\right), \dot{q}\left(t\right)\right)\dot{q}\left(t\right) - g\left(q(t)\right)\right] \tag{3.10}$$

$$\dot{x}\left(t\right) = \left[\begin{array}{c} \dot{q}\left(t\right) \\ \ddot{q}\left(t\right) \end{array}\right] = \left[\begin{array}{c} 0_{N\mathrm{x}1} \\ Mq(t)^{-1}\left(\tau - \tau_d\right) \end{array}\right] - \left[\begin{array}{c} -\dot{q}\left(t\right) \\ Mq(t)^{-1}C\left(q, \dot{q}\right) \end{array}\right] \tag{3.11}$$

$$y = \left[\begin{array}{cc} I & 0 \end{array}\right]\left[\begin{array}{c} q\left(t\right) \\ \dot{q}\left(t\right) \end{array}\right] \tag{3.12}$$

3.2 Path/Geometric Constraints

To realize the overall objective of collision avoidance in real-time, the dynamic equations of the robot manipulator model which specifies its motion interacts with the geometric properties of an obstacle in its workspace. This geometric structure of the interacting links is an implied additional constraint with the prior differential constraint of the dynamics.

3.2.1 Dynamic Obstacles

Representing obstacle points in the Cartesian coordinate of the robot with time varying properties as:

$$obs_i = \left[obs_i^x(t), obs_i^y(t), obs_i^z(t)\right] \in \mathbb{R}^3 \text{ for } t \in [t_i, t_f]$$

where $obs_i^{(x,y,z)}(t)$ represents the obstacle Cartesian coordinate location with time varying properties, hence, the number of obstacle N_{obs} and its location $obs_i^{(x,y,z)}(t)$ changes with respect to time $N_{obs}(t) = [L, 2, ..., N_{obs}(t)]$.
Also, a set defining the full geometry of all robot interacting links as $E_l(t)$ which is also time varying as:

$$E_L \triangleq \{p_L | p \in \mathbb{R}^3, E_L(p_L, q(t))\} \leq 0$$

representing the space enclosed region were the L_{th} link of the robot is contained.
To capture this time varying characteristics of the obstacle coordinate, the most recent obstacle coordinate needs to be communicated at runtime for the problem setup to compute collision-free trajectories. As a result, the receding horizon control setup is used to implement this fractional solution instance.
This is achieved through the bi-directional communication of the robot most recent environment variables (obstacle location and robot pose) for computations of sub-optimal trajectory and solution to the optimal control solver to ensure the integrity of the trajectory as collision-free at each solution time-step.

3.2.2 Representing the Link Geometry: Minimum Enclosing Ellipsoids

The robot needs some information about its environment to properly manipulate itself for obstacle avoidance. Position coordinates of any object in the Cartesian plane is sufficient as long as the work envelope of the interacting links can defined and the robot knows where it is at any time instance.

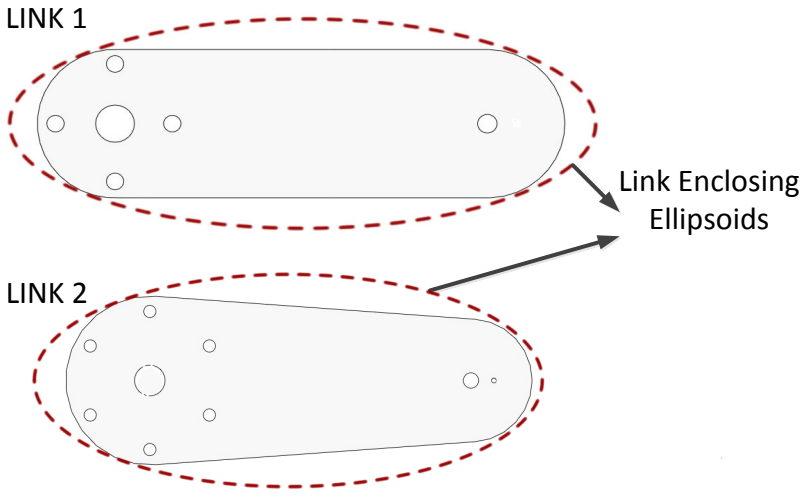

FIGURE 3.1: Illustration of the Minimum area ellipsoids enclosing the robot links

Depending on the shape of the manipulator links, standard equations with a minimum enclosing volume property to represent it geometric structure may be defined. Both

links of the robot investigated in this work are not of any standard shapes. A minimum enclosing ellipsoid is used to encapsulate the links and hence define its kinematic relationship. This formulation was adopted from the implementation in [64–66]. More formal derivations are found in [67, 68]. The respective links entire volumes are confined in these ellipsoids and represented by the standard equation of an ellipsoid.

The center axis of the minimum enclosing ellipsoid formulation at each individual successive link defined as $p_{L=1(x,y,z)}, p_{L=2(x,y,z)}, \ldots, p_{L=L_{th}(x,y,z)}$ for links $L = 1, 2, ..., N$. For example, in a planar configuration, with c_i denoting the distance from the center of the l'th joint to the center of the ellipse, the location coordinate of the center of ellipsoid may be represented as follows:

$$\begin{bmatrix} p_L(x) \\ p_L(y) \end{bmatrix} = \sum_{L=1}^{N} \begin{bmatrix} \cos(q_{L-1}) \\ \sin(q_{L-1}) \end{bmatrix} + c_i \begin{bmatrix} \cos \sum_{l=1}^{N} q_L \\ \sin \sum_{l=1}^{N} q_L \end{bmatrix} \tag{3.13}$$

With each link, $\phi_l = \sum_{l=1}^{N} q_{l_{th}}$, obstacle coordinate information derived from the work space environment is defined as $[obs^{(x)}, obs^{(y)}, obs^{(z)}]$, the smallest ellipsoid enclosing each link $E_{L=1}, E_{L=2}, ..., E_{L=N}$ is given as:

$$E_L = \frac{\left[(obs^{(x)} - p_L(x))cos\phi_L - (obs^{(y)} - p_L(y))sin\phi_L\right]^2}{{a_L}^2} + \frac{\left[(obs^{(y)} - p_L(y))cos\phi_L - (obs^{(x)} - p_L(x))sin\phi_L\right]^2}{{b_L}^2} - 1 = 0 \tag{3.14}$$

3.3 Collision-free Optimal Trajectory Generation under Dynamic Environments

3.3.1 The Cost Functional

The path along the functional to be minimized is the vector values quadratic cost $J = C(q(t), \tau(t))$. The optimal solution of this problem is the solution of a cost functional of infinite time dimension given by:

$$J(q(t), \tau(t)) = \int_{t_i}^{\infty} C(q(t), \tau(t))dt \tag{3.15}$$

For a possible scaling of a time-step dependent control scheme, the finite OCP can be represented as:

$$J(q(t), \tau(t)) = C\left(q\left(t_f\right), \tau\left(t_f\right)\right) + \int_{t_i}^{t_f} C\left(q\left(t\right), \tau(t)\right) dt \;\; t \in [t_i, t_f] \tag{3.16}$$

If $q(t) \in \mathbb{R}^N$ and $\tau(t) \in \mathbb{R}^N$ are the state and control vectors along $t \in [t_i, t_f]$ associated with minimizing the cost. Assuming the element $\tau \in C_{\infty}^{n.\tau}[t_i, t_f]$ is space of all Lebesgue-measurable functions with absolute values of its entity bounded on $[t_i, t_f]$ implying there exist control trajectories translating to the unique state trajectory in the space where the system is controllable.

The optimal control and state vector $\tau^* = {\tau_0}^*, {\tau_1}^*, {\tau_2}^*, {\tau_3}^*, \ldots, {\tau_{N-1}}^*$ and $q^* = {q_0}^*, {q_1}^*, {q_2}^*, {q_3}^*, ..., {q_N}^*$ generated from the minimization of the chosen performance index above with respect to the constraint on the system equation is the so called optimal solution.

The vector valued function for minimum energy prescribe for the manipulator links is defined as $\|\tau_L\left(t\right)\|^2$. The problem may be further constrained by specifying a desired time-trajectory for the robot to follow in addition to the task to move from an initial to final poses hence a trajectory tracking error in the robot taskspace given as: $T_r = [x - x_r \;,\; y - y_r]$ which is the difference between the manipulators present robot state $[x, y]$ and a predefined desired state $[x_r, y_r]$. Weighting factor for the control variable W_u and tracking error W_e are incorporated to scale the performance of the solution as a coordinate tracking or minimum energy problem.

The cost functional to be minimized is given as:

$$J(q(t), \tau(t)) = \frac{1}{2} \int_{t_i}^{t_f} \left\{ W_u \left(\|\tau_1(t)\|^2 + ... + \|\tau_i(t)\|^2 \right) + W_e \|T_r(t)\|^2 \right\} dt \quad t \in [t_i, t_f] \tag{3.17}$$

Defining the norm (Euclidean) as $\|\tau\| = \left(\tau^{\mathrm{T}}\tau\right)^{1/2}$ which assigns a scalar $\|\tau\|$ to every $\tau \in \mathbb{R}^N$ such that $\|\tau\| \leq 0$ for all $\tau \in \mathbb{R}^N$ and $\|\tau\| = 0$ if and only if $\tau = 0$

3.3.2 Limits on Control and State Variables

Imposed constraint sets on the state and controls variables are in order to implement the solution on an experimental setup. Limits on the amount of force a motor may disperse to the links and stable motion a state may specify creates a min-max boundary limit on them. This may be specified as follows:

$$\left.\begin{array}{l} q_{lower} \leq q\left(t\right) \leq q_{upper} \\ \tau_{lower} \leq \tau\left(t\right) \leq \tau_{upper} \end{array}\right\} t \in [t_i, t_f] \tag{3.18}$$

3.3.3 Initial and Final Boundary Conditions

The Initial and final configuration boundary conditions of the optimization state and event variables may also be specified as follows:

$$\left.\begin{array}{l} q^0 = \left(a_{lower}, ..., a_{upper}\right), q^T = \left(a_{lower}, ..., a_{upper}\right) \\ \dot{q}^0 = \left(b_{lower}, ..., b_{upper}\right), \dot{q}^T = \left(b_{lower}, ..., a_{upper}\right) \end{array}\right\} t \in [t_i, t_f] \tag{3.19}$$

3.3.4 Finite Horizon Optimal Control Problem

Problem 1: The overall problem can be summarized as follows:

$$J(q(t), \tau(t)) = \frac{1}{2}\int_{t_i}^{t_f} \left\{ W_u \left(\|\tau_1(t)\|^2 + ... + \|\tau_i(t)\|^2 \right) + W_e \|T_r(t)\|^2 \right\} dt \quad t \in [t_i, t_f] \tag{3.20}$$

$$subject\ to:$$

$$\dot{x}\left(t\right) = \begin{bmatrix} \dot{q}\left(t\right) \\ \ddot{q}\left(t\right) \end{bmatrix} = \begin{bmatrix} 0_{n\text{x}1} \\ Mq(t)^{-1}\left(\tau - \tau_d\right) \end{bmatrix} - \begin{bmatrix} -\dot{q}\left(t\right) \\ Mq(t)^{-1}C\left(q, \dot{q}\right) \end{bmatrix} \tag{3.21a}$$

$$obs_{lower}^{(x_l, y_l)} \leq \left[q\left(t\right), \tau\left(t\right), t\right] \leq obs_{upper}^{(x_l, y_l)} \; t \in [t_i, t_f] \tag{3.21b}$$

$$\left.\begin{aligned} q_{lower} \leq q\left(t\right) \leq q_{upper} \\ \tau_{lower} \leq \tau\left(t\right) \leq \tau_{upper} \end{aligned}\right\} t \in [t_i, t_f] \tag{3.21c}$$

$$\left.\begin{aligned} q^0 = \left(a_{lower}, ..., a_{upper}\right), q^T = \left(a_{lower}, ..., a_{upper}\right) \\ \vdots \\ \dot{q}^0 = \left(b_{lower}, ..., b_{upper}\right), \dot{q}^T = \left(b_{lower}, ..., a_{upper}\right) \end{aligned}\right\} t \in [t_i, t_f] \tag{3.21d}$$

A pair of $(\tau(t), q(t))$ is feasible if all the constraint sets above are fulfilled in addition, a feasible pair (τ^*, q^*) is also optimal if

$$J\left(\tau, q\right) \geq J\left(\tau^*, q^*\right) \tag{3.22}$$

For all sets of feasible pairs $(\tau(t), q(t))$. Along the optimization path, the functions $\tau^*(t)$ and $q^*(t)$ are known as the respective optimal control and optimal state trajectory functions.

3.4 Solving the Problem

3.4.1 Receding Horizon Control (RHC)

A one shot solution to the problem above would be sufficient if all the values of the optimization variables were known for the entire time span $[t_i, t_f]$ at the initial stage of computing the solution. This is not the case as the obstacle location are time varying hence the problem setup will need to be conditioned to capture and accommodate these dynamic time varying constraint set.

Based on the highlighted properties of the obstacle constraint set, the OCP is restructured using the receding horizon control scheme in order to capturing the dynamic time-varying nature of the obstacle(s). This allows for an iterative computation of sub-optimal solutions in successive time intervals of the choice of a safe control horizon.

The prediction horizon p_h is the time frame which captures the initial to terminal state of the system in which time span, successfully collision-free optimal trajectory may be planned.

A control horizon c_h may be designed as a *safe enough time range* in which the robots single time-step sub-optimal solutions (set of optimal states/control) suffices as collision-free and thus re-initializing the new initial state for next time-step problem to be solved at the terminal state of the control horizon. At the implementation stage, the control horizon is computed based on some collision-free check rules on the robot realtime controller.

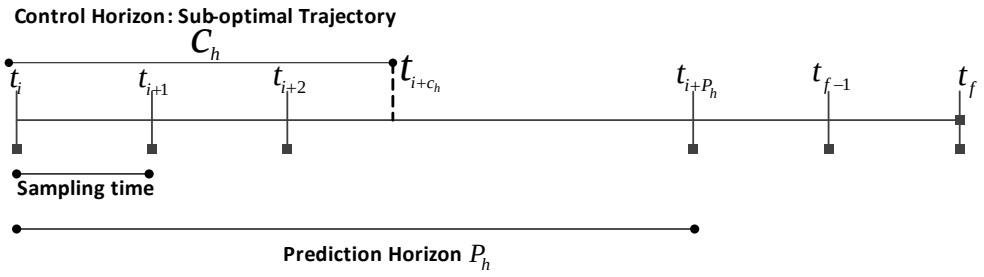

FIGURE 3.2: Receding Horizon control Framework

Hence with t_i as the initial time step of the OCP, the solution set (set of sub-optimal control/state trajectory) is applied across the time step t_i+c_h; a fraction of the prediction horizon. The choice of this control horizon is informed based on the robot coordinate location with respect to the dynamic obstacle coordinate ensuring a safe enough motion span at evolving time steps of the robot.

STEPS

1. Compute the entire set of optimal control inputs within a fixed prediction horizon $\tau_i^* = [\tau_1^*, ..., \tau_{t_i+P_h}^*]$ based on the most recent obstacle location and robot state.

2. Implement a fraction of the sub-optimal controls based on the robot present state and the obstacle most recent coordinate location at solution time to collision-free motion spanning $\tau_i^* = [\tau_1^*, ..., \tau_{t_i+C_h}^*]$ (control horizon).

3. Reinitialize the RHC problem with the time step t_{i+c_h} as the new initial time t_i and iterate through step 1. The problem terminates at the point where the final desired robot state is reached which is the point where the control horizon is exhausted i.e. $c_h = t_f$ or where the control and prediction horizon have the same terminal state.

Problem 2 The overall Receding Horizon OCP is stated as:

$$J(q(t),\tau(t)) = \frac{1}{2}\int_{t_i}^{P_h}\left\{W_u\left(\|\tau_1(t)\|^2 + ... + \|\tau_i(t)\|^2\right) + W_e\|T_r(t)\|^2\right\}dt \quad t \in [t_i, P_h] \tag{3.23}$$

subject to:

$$\dot{x}\left(t\right) = \begin{bmatrix} \dot{q}\left(t\right) \\ \ddot{q}\left(t\right) \end{bmatrix} = \begin{bmatrix} 0_{n\text{x}1} \\ Mq(t)^{-1}\left(\tau - \tau_d\right) \end{bmatrix} - \begin{bmatrix} -\dot{q}\left(t\right) \\ Mq(t)^{-1}C\left(q,\dot{q}\right) \end{bmatrix} \tag{3.24a}$$

$$obs_{lower}^{(x_l,y_l)} \leq \left[q\left(t\right),\tau\left(t\right),t\right] \leq obs_{upper}^{(x_l,y_l)}\ t \in [t_i, P_h] \tag{3.24b}$$

$$\left.\begin{aligned} q_{lower} \leq q\left(t\right) \leq q_{upper} \\ \tau_{lower} \leq \tau\left(t\right) \leq \tau_{upper} \end{aligned}\right\} t \in [t_i, P_h] \tag{3.24c}$$

$$\left.\begin{aligned} q^0 = \left(a_{lower},...,a_{upper}\right), q^T = \left(a_{lower},...,a_{upper}\right) \\ \vdots \\ \dot{q}^0 = \left(b_{lower},...,b_{upper}\right), \dot{q}^T = \left(b_{lower},...,a_{upper}\right) \end{aligned}\right\} t \in [t_i, P_h] \tag{3.24d}$$

3.4.2 Discrete Time Optimal Control Problem

A discrete-time OCP can be derived from a time discretization of the continuous time OCP in problem 1 above and stated as follows:

$$J\left(q_i,\tau_i\right) = \phi\left(q_N,\tau_{N-1}\right) + \sum_{i=0}^{N-1} C_i\left(q_i,\tau_i\right) \tag{3.25}$$

subject to:

$$D_s \triangleq \begin{pmatrix} f_o(q_0, \tau_0) - q_1 = 0 \\ f_1(q_1, \tau_1) - q_2 = 0 \\ \vdots \\ f_N(q_N, \tau_{N-1}) - q_N = 0 \end{pmatrix} \tag{3.26}$$

D_s = Systems discretized dynamics at each time steps k_i from $k_1, k_2, ..., k_{N-1}$. And with boundary conditions on the state and control variables to put realism into context $q_{\min} \geq q_{i \to N} \leq q_{\max}, \tau_{\min} \geq \tau_{i \to N-1} \leq \tau_{\max}$.

This is in summary a nonlinear programming problem.
Defining a mapping function written in terms of the control parameters $\varphi_i(\tau)$ based on the systems dynamics 3.9 which relates the state and control trajectory as, $q_i = (\varphi_i(\tau))$ which translates any given control trajectory to its corresponding unique state trajectory. The corresponding Lagrangian representing the discretized system above is given as:

$$J(\varphi_i(\tau), \tau_i, \tilde{\lambda}) = C_N(\varphi_N(\tau)) + \sum_{i=0}^{N} \tilde{\lambda}_i^T \left[f(\varphi_i(\tau), \tau_i) - \sum_{k=0}^{N} D_s \right] \tag{3.27}$$

$$\tau = [\tau_0, \tau_1, \ldots, \tau_{N-1}]^T$$
$$\varphi(\tau) = [\varphi(\tau)_1, \varphi(\tau)_2, \ldots, \varphi(\tau)_N]^T$$
$$\tilde{\lambda} = \tilde{\lambda}_1, \tilde{\lambda}_2, ..., \tilde{\lambda}_N$$

3.5 Optimality Conditions of the Discretized Problem

Where H is the Hamiltonian function defined at all the discrete points along the path as: $H_i(q_i, \tau_i, P_{i+1}) = C_i(q_i, \tau_i) + P_{i+1} f_i(q_i, \tau_i)$; discretized states $q_i = \bar{q}_i$ and discretized controls $\tau_i = \bar{\tau}_i$ recalling the relationship between the state and controls defined above as $q_i = (\varphi_i(\tau))$.

1st order condition: $\nabla J(\tau^*) = 0$ for $\tau^* = (\tau_0^*, \tau_1^*, \tau_2^*, ..., \tau_{N-1}^*)$ to be the local minimum control trajectory and with corresponding state trajectory uniquely specified by the system equation consequently as $q^* = (q_0^*, q_1^*, q_2^*, ..., q_N^*)$, then $\nabla_{\tau_i} H_i({q_i}^*, {\tau_i}^*, {P_{i+1}}^*) = 0, i = 0, ..., N-1$ where the costate vectors $P_1, ..., P_n$ are derived from the adjoint equation ${P_i}^* = \nabla_{q_i} H_i({q_i}^*, {\tau_i}^*, {P_{i+1}}^*) = 0;\ \imath = 0, 1, ..., N-1$ with terminal conditions on the state constraints as ${P_N}^* = \nabla C_N({q_N}^*)$.

3.6 Discretizing the Problem

As earlier expressed, a time discretization setup needs be engaged to transcribe the OCP into a NLP which in turn can be solved for local minimums.

The suitability of the collocation and Discrete mechanics for optimal control methods are based on some mechanical properties of the KCM system. The problem can be views as a constrained optimal control system with highly nonlinear system dynamics. The Collocation method utilizes the combined action of high order Legendre polynomials and quadrature rules to approximate the systems governing ODE's and compute integral terms respectively. This ensures high degrees of accuracy in the solutions of the resulting NLP.
Furthermore, the discretization process in the Collocation methods especially the global methods handle nonlinearities quite well and captures the dynamics fully by concentrating more collocation points in areas of high activity resulting in even better solutions with lower approximation errors hence higher accuracy.

The DMOC method is also a good candidate for this class of problems as it's geometry preserving discretization process keeps the geometric structure of the mechanical system in terms of energy conservation and physical symmetry ensuring momentum conservation (symplectic property) and good energy behavior for equal time steps. This makes the problem immune to some ill-conditioning in term of stable physical geometry hence the solutions have been proven to result in very accurate solutions. It also gives a discretization scheme of the system in the configuration space which makes the resulting NLP quite exact as well and its the algorithm is quite straight forward.

Both methods; the DCM and DMOC discretize simultaneously the state and control variables which in turn increases the size of the inequality constraints resulting in a larger dimension NLP. This is taken care of due to the high sparsity nature in the Jacobian matrices. Solutions to the resulting NLP can still be found at considerable short time span feasible for real-time applications.

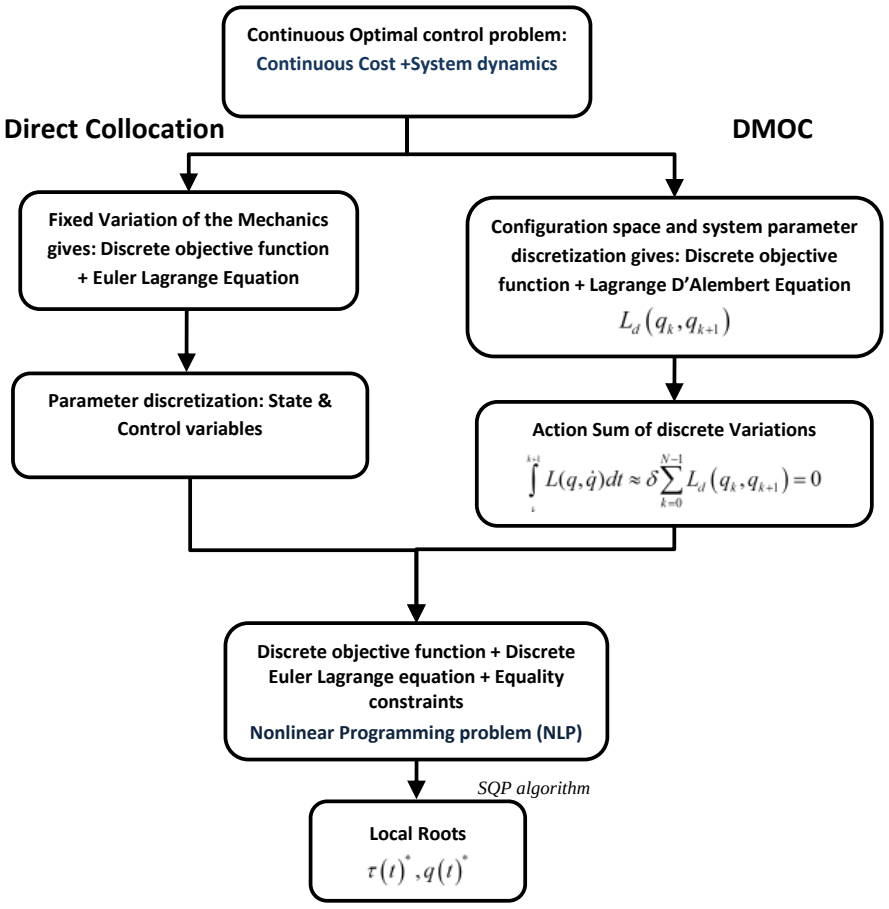

FIGURE 3.3: Overall transcription process of an OCP to NLP by DMOC and DCM

3.6.1 Direct Collocation Method (DCM)

The collocation method is a transcription process which discretizes a continuous-time optimal control problem into a finite dimensional Nonlinear programming one. The idea stems from the minimization of approximate errors in the functional over the entire domain, however in this case, the approximation is done at interval points called collocation nodes/points/grid/intervals over the domain of the problem along the optimization path [45]. There are a good number of established collocation methods used for direct transcription and what makes it desirable for this class of optimal control problem is the availability of adjoint costate estimates that relate the Lagrange multiplier of the discretized problem to the adjoint vector whose variables are evaluated to find global solutions at collocation points [46] as shown in the setup below.

The time discretization method parameterizes both the state and control variables as optimization variables. This results into a larger NLP because of the use of both the

state and control variables. This however is acceptable because even thou the constraints sets are more, the derivatives of this constraints i.e. the Jacobian of the system have many zero values at the nodes. This sparsity property of the Jacobian accounts for why the eventual NLP can still be achieved in reasonable time in fact, this method has shown to assist converge to optimal of OCP at a spectral rate and possible real-time optimization.

Whichever it is the Collocation method one adopts, the process and end-result to a problem of this nature:

* Discretize the Optimal control problem into a NLP by time discretization into a grid of n intervals.
* Approximate the states x and control inputs u as piecewise polynomials at each successive node.
* Collocate the midpoint of successive intervals within the number of discretized time grid points corresponding to the value of the state and control at each end grid point. An intuitive method will be to integrate the differential equation over a one time-step and then approximate the integral on the right by a finite sum.
* Solve the resulting finite dimension approximation of the original control problem.

Considering a nonlinear dynamic system whose dynamics is represented similar to the nonlinear state space (SS) *problem 1* as follows:

Nonlinear State Space dynamic equation: $\dot{x}(t) = f(x(t), u(t))$.

M inequality constraint set: $M(x(t), u(t), t) \geq 0\ U_i \subset \mathbb{R}^N,\ i = 0, 1, ..., N-1$.

State and control bounds: $r_{\min} \leq r(x(t), u(t), t) \leq r_{\max}$.

Initial conditions: $b_{\min} \leq r(u(t_0), t_0, x(t_f), t_f) \leq b_{\max}$.

With corresponding lower and upper boundaries as min/max values at the collocation points.

The objective is to find a vector u(t) that minimizes a cost functional representing a systems objects to be optimized in the form:

$$J(x(t)u(t)) = C(x(t_f), u(t_f), t_f) + \int_{t_i}^{t_f} C(x(t), u(t), t)dt \tag{3.28}$$

and the discrete equivalent given as:

$$J(x_d(t)u_d(t)) = \sum_{i=0}^{N} C(x_i, u_i, t_i) \tag{3.29}$$

Generally, the initial and final time is specified or may be non-static based on the optimization solution for the state variables.

The direct collocation method prescribes approximate numerical solutions to the problem above by discretizing the time set of spaced time grid $(N-1)$ samples known as nodes with respect to the state and control trajectory within this collocation points (nodes) are approximated by suitable piecewise functions of high order polynomials which gives the corresponding high approximate finite dimensional NLP to be solved.(Note: for the problem investigated, discretizing the control variable set is needed based on the problem formulation).

3.6.1.1 Collocation Implementation Steps

Following the algorithm setup in scheme in [69], the approximation scheme for computing the collocation points and ensuring the optimal solution derived at each node is satisfied and the constraint set imposed on the objective function based on the dynamics of the system is not violated. For the problem given above, say the discretized time interval be carefully chosen and specified as

$$0 \leq t_1 < t_2 \ \ldots \ < t_N = t_f \tag{3.30}$$

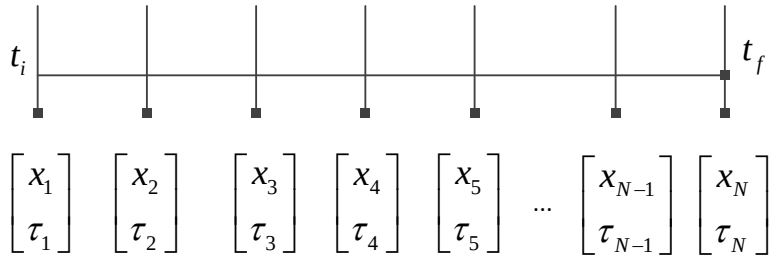

FIGURE 3.4: Discrete state and control structure at each node

For control and state variables at the grid point $t_i = 1, 2, ..., N$ and final time t_f.

The parameter employs a solution vector of the form:

$$S = [u(t_1), ..., u(t_N);\ x(t_1),\ ...\ , x(t_N);\ t] \quad \mathbb{R}^{N\text{x}M} \tag{3.31}$$

The controls vector are chosen as piecewise linear interpolation functions between $u(t_i)$ and $u(t_{i+1})$. The states is chosen likewise for $x(t_i)$ and $x(t_{i+1})$ with sample time $s = [t_i, t_{i+1}] = [-1, 1]$ as the solution time at each node, the dynamic constraint is rewritten as:

$$\dot{x}_i(s) = f(x(s), u(s), s)$$
$$At\ s = t_{i+1} - t_i$$

Hence have mesh/grid point for collocation interpolation is given as:

$$\left.\begin{array}{l} t_i\,[i = 1, ..., N] \\ t_i\,[i = 1, ..., N-1] \end{array}\right\} \begin{array}{l} \text{for States} \\ \text{for Controls} \end{array}$$

These points are popularly referred to as the Gauss-Lobatto points.

A piecewise polynomial (which is continuous differentiable) is used to estimate the objective function between each intermediate point i.e. $p(t_i)$ and $p(t_{i+1})$.
The proper approximation to the costate leads to a set of KKT conditions identical to the discretized form of the 1st order optimality conditions stated in 3.5 earlier.

This local interval center approximation is done using the formulas stated as follows, with N being the number of collocation points with state and control;

$$x_{(i,c)} = \frac{1}{2}(x_i - x_{i+1}) + \frac{T}{N}(x_i + x_{i+1}) \tag{3.32}$$

$$u_{(i,c)} = \frac{1}{2}(u_i - u_{i+1}) \tag{3.33}$$

The choice of this piecewise polynomial may vary and they are classed into two: local/direct and global discretization.

The quadrature rule used to approximate the integral part of the cost function in computing the optimality conditions for the controls at the nodes are the same for either classes be it local or global discretization and is known as the GaussLobatto quadrature rule. It consists of a weighted sum of the function values at the discretization nodes.

The approximate function is computed with the property $\hat{x}_i(s) = f(x(s), u(s), s)$ at $s = t_i \, and \, t_{i+1}$ as follows:

States approximates $\hat{x}_i(t) = \sum_{k=0}^{N} c_i \frac{t - t_i^j}{s_i}, \quad t_i \leq t \leq t_{i+1}; \quad i = 1, 2, ..., N$

Controls approximates $\hat{u}_i(t) = u(t_i) + \frac{t - t_i}{s}(u(t_{i+1}) - u(t_i)); \quad i = 1, 2, ..., N-1$

3.6.1.2 Local and Global discretization

The **local discretization** is a static meshing operation which consists of using *the Trapezoidal method, Central difference method or Hermit Simpsons method* as the approximation function for the associated optimal control problem cost function within successive nodes. Usually, the resulting Jacobian and Hessian are sparser especially as the number of collocation point's increase.

The approximation functions for the discretization are computed as follows:

For the ***trapezoidal method*** $c_i = x(\tau_{i+1}) - x(\tau_i) - \frac{s}{2}(f_i + f_{i+1})$

Central difference method

$$\begin{aligned} D_{0,0} &= -1/s_0, D_{0,1} = 1/s_0 \\ D_{i-1,i} &= 1/(s_i + s_{i-1}), D_{i-1,i-2} = -1/(s_i + s_{i-1}) \\ D_{N,N-1} &= -1/s_{i-1}, D_{N,N} = 1/s_{i-1} \end{aligned} \tag{3.34}$$

Hermit-Simpsons method

$$\begin{aligned} c_i &= x(\tau_{i+1}) - x(\tau_i) - \frac{s}{6}\left(f_i + 4\bar{f}_i + f_{i+1}\right) \\ \bar{f} &= f\left[\bar{x}_{i+1}, \bar{u}_{i+1}, \tau_i + \frac{s}{2}\right] \\ \bar{x}_{i+1} &= \frac{1}{2}x(\tau_i) + x(\tau_{i+1}) + \frac{s}{8}(f_i - f_{i+1}) \end{aligned} \tag{3.35}$$

And the costate estimate for local discretization is given as $\lambda(t_{i+1/2}) = \frac{\hat{\lambda}_i}{2s_i}, \mathrm{i} = 0, ...N-1$ The 3 local discretization methods described above creates a regular discretization for a system of ODE's as shown below. This may be sufficient and even advantageous depending on the structure of the problem in question with respect to the momentum flow along the path of discretization and linearity of the systems dynamic equations. The approximation errors in the solutions of the NLP generated through local discretization methods are large when implemented for systems with highly nonlinear dynamics.

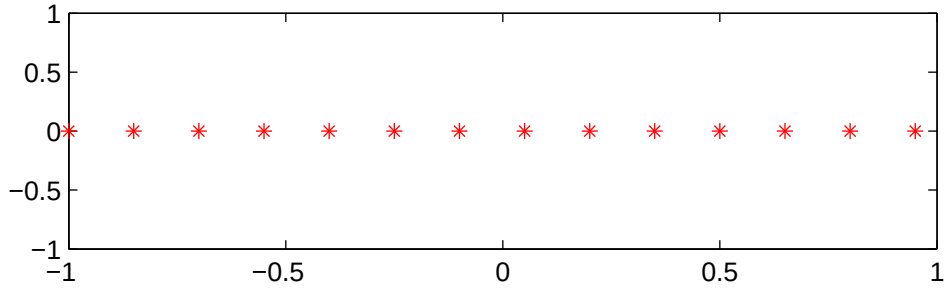

FIGURE 3.5: Illustration of the static nodes feature for N=20

The **global discretization** methods is a dynamic interpolation method for mesh refining which is popularly referred to as the moving mesh and are generally known as pseudo-spectral methods. They comprise of using *the Legendre polynomials, Chebyshev*

polynomials or Gauss polynomial functions as approximate functions for the cost function being minimized similarly as in the case for the local discretization scheme.
It is primarily designed for systems with highly nonlinear dynamics equations.
This method uses a sample time $s = [-1, 1]$ and collocation are done at orthogonal collocation points. It has an exponential rate of convergence and the preceding interpolation points are quite accurate based on the algorithms essentially because the grid points are more concentrated at points in between the node with greater activity resulting into a function with better approximates hence higher accuracy. Another advantage of the global discretization scheme is that it increases the sparsity ratio of the Jacobian matrices in the NLP due to the irregular time discretization intervals. This in turn allows for faster root solutions to be computed using sparse nonlinear programming algorithms/-solvers. A typical structure of an irregular mesh setup is shown below. It typically gives less approximation errors for systems that are highly nonlinear compared to the local methods.

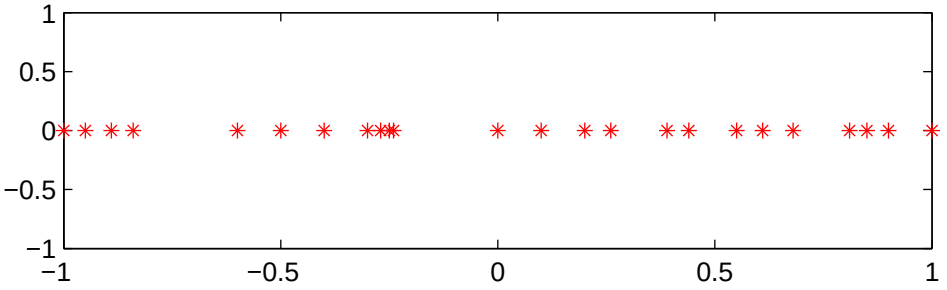

FIGURE 3.6: Illustration of the moving mesh/dynamic nodes feature for N=20

Using Legendre polynomials and numerical quadrature at the Gauss Lobatto points, with j specifying the order of the polynomial used for approximating the state (usually cubic) is given as follows:

$$\begin{aligned}
c^i_{j=0} &= x(t_i), \\
c^i_{j=1} &= f_i, \\
c^i_{j=2} &= -3x(t_i) - 2s f_i + 3x(t_{i+1}) - s_i(f_{i+1}), (quadratic) \\
c^i_{j=3} &= 2x(t_i) + s_i f_i - 2x(t_{i+1}) + s_i(f_{i+1}), (cubic) \\
&\vdots \\
c^i_{j=j^{th}} &= (j^{th}\ order\ polynomials)
\end{aligned} \tag{3.36}$$

The carefully chosen approximates above already fulfill the solution with respect to the constraints sets at the nodes t_i and the only other constraints left are the

- Collocation constraints at the center

$$f(\hat{x}(t_{c,i}), \hat{\tau}(t_{c,i}), t_{c,i}) - \hat{\dot{x}}(t_{c,i}) \leq 0, \quad i = 1, ..., N \tag{3.37}$$

- Inequality constraints at the grid points

$$M(\hat{x_i}(t), \hat{\tau_i}(t), t_i) \leq 0, \quad i = 1, ...N \tag{3.38}$$

- Initial and end point constraints in the range of the integral solutions.

The costate estimate for the sampling time [-1,1] of the piecewise polynomials are given by

$$\lambda(u_i) \approx \lambda^N(u_i) = \sum_{i=0}^{N} \lambda(\tau_i) x_i(\tau), \quad s \in [-1, 1] \tag{3.39}$$

More emphasis on choice of choosing suitable polynomial functions for different classes of problems for discretization can be found in [45, 69, 70].

3.6.1.3 Necessary first Order Optimality Conditions

The necessary first Order optimality conditions for the solution of the discretized OCP takes the form:

$$\frac{\partial C}{\partial u_i} = 0, \quad i = 1, ..., N \tag{3.40a}$$

$$\frac{\partial C}{\partial x_i} = 0, \quad i = 1, ..., N \tag{3.40b}$$

$$\frac{\partial C}{\partial \lambda_i} = 0, \quad i = 1, ..., N - 1 \tag{3.40c}$$

$$M(x_i, u_i, t_i) = 0 \tag{3.40d}$$

3.6.2 Discrete Mechanics for Optimal Control (DMOC)

In the collocation method, a combination of the cost function and the equation of motion (Euler-Lagrange equation), is time discretized along selected points on the optimization

path to be minimized. These resulted to a corresponding NLP which is further solved using SQP for the optimal trajectory. The idea is backed by minimizing the resulting approximation error at the end of each node.

The discretization brought about using discrete geometric mechanics is somewhat similar to the direct collocation method. In this case, rather than approximating the equations of motion coupled with the objective function as in the DCM, a discretization of the variational formulation of the Lagrange equations directly using the systems geometric arguments is carried out. The strength of this method comes about by using *variational time integrators* which are symplectic in nature in order to preserve the geometry of the problem and hence energy conservation and stability form of the mechanical system [55, 56].

A straight forward way of discretizing the problem by adopting the formulation in the continuous realm gives a discrete notion equations and OCP and in turn a finite dimensional NLP problem is still the end result of the scheme just as is with the DCM.

This is done by applying the discrete variations of the mechanical systems parameters to first describe the system in question. The resulting formulation is known as the Lagrange-d'Alembert principle. Then direct discretization is done using these Variational integrators which have the property of nearly preserving the structure of the discrete energy and momentum relating to a symplectic form of the Lagrangian flow.

The control effort of the system along the discrete path is cleverly choosing as fiber preserving discrete mappings steps to keep the consistent symplectic structure of the dynamics in the case of a forced Lagrangian system [53].

The problem is transformed into an equality constrained NLP through the finite parameterization of the state and control variables and solved using an NLP algorithm.

3.6.2.1 Implementation Steps

The continuous case OCP and with dynamic and inequality constraint as stated in *Problem 1* is invoked here as our base problem.

The variation of the action i.e. the Lagrangian L along a path $q(t) \in Q$ for a finite time is given as:

$$\int_{t_i}^{t_f} L(q(t), \dot{q}(t))dt \tag{3.41}$$

to move the system from an initial $(q^{t_i}, \dot{q}^{t_i})$ state to final state $(q^{t_f}, \dot{q}^{t_f})$ The above is a self- conserving momentum system. For a forced system, the corresponding equation can be stated as:

$$\delta \int_{t_i}^{t_f} L(q(t), \dot{q}(t))dt + \int_{t_i}^{t_f} f_L(q(t), \dot{q}(t), \tau(t)) \cdot \delta q(t)dt = 0 \tag{3.42}$$

Popularly referred to as the *Lagrange-d'Alembert* principle which states that for a time steps $k_i, k_{i+1}, k_{i+2}, ..., n$ for all variations of $\delta q(t)$ with $\delta q(t_i) = 0$ and $\delta q(t_f) = 0$

Following the DMOC formulation in [57]and[53], the successive time shift is used to discretize the functions by changing the equation into a first order equation. This is done accordingly. The time step is given as: $h = k_{i+1} - k_i$

Continuous Setting	**Discrete Setting**
	$q = q_k$
$\dot{q}(t) = \frac{dq}{dt}$	$\dot{q}(t) = q_{k+1} - q_k$
$\dot{v} = \ddot{q}(t) = \frac{d^2q}{\partial t^2}$	$\dot{x} = x_{k+1}$
	$\tau(t) = \tau_{\frac{k+1}{2}}$

Time integrator is given as:

$$v_{k+1} = v_k - h(A, q_k)$$
$$q_{k+1} = q_k + h(v_{k+1})$$

This integrator scheme is known as the symplectic Euler method.

Discretizing the Lagrange equation of the system, the resulting formulation results into

$$\delta \sum_{k=0}^{N-1} L_d(q_k, q_{k+1}) \approx \int_{k}^{k+1} L(q, \dot{q})dt \tag{3.43}$$

The intermediate control force at the discretized points (grid) $\tau_i = \tau_1, \tau_2, \tau_3..., \tau_{n-1}$ moving the system from the corresponding initial state to the end point $q_i = q_1, q_2, q_3, ..., q_N$ is specified as $f_L(u_1, ..., u_N)$ Taking two set of discrete Lagrange control forces and its coupled sum for balancing of force dissipation, It gives:

$$f_{L_d}(q_k, q_{k+1}).(\delta q_k, \delta q_{k+1}) = f_d^-(q_k, q_{k+1})\delta q_k + f_d^+(q_k, q_{k+1})\delta q_{k+1} \tag{3.44}$$

Discrete Lagrange Equation:

$$\sum_{k=0}^{N-1} L_d(q_k, q_{k+1}) + \sum_{k=0}^{N-1} [f_d^-(q_k, q_{k+1})\delta q_k + f_d^+(q_k, q_{k+1})\delta q_{k+1}] = 0 \tag{3.45}$$

Taking fixed end variations of the above equation i.e. partial derivatives of each term with respect to the discretized steps for a specified number of nodes along the problem time duration and the sum of all the actions results in the discrete Euler-Lagrange equations equivalent to the equation of motion of the system given as:

$$\sum_{k=0}^{N-1} [D_1 L_d(q_k, q_{k+1}) + D_2 L_d(q_{k-1}, q_k)]\cdot\delta q_k + \sum_{k=0}^{N-1} [f_d^-(q_k, q_{k+1})\delta q_k + f_d^+(q_k, q_{k+1})\delta q_{k+1}] = 0 \tag{3.46}$$

with D_1 and D_2 denoting the partial derivatives with respect to q_k and q_{k+1} respectively. The resulting equation gives the corresponding motion equation of the system which is the Discrete Euler Lagrange Equation (DELE)

Discrete Lagrange-d'Alembert equation

$$\begin{gathered} D_1 L_d(q_k, q_{k+1}) + D_2 L_d(q_{k-1}, q_k) + f_d^-(q_k, q_{k+1}, \tau_k) + f_d^+(q_k, q_{k+1}, \tau_{k-1}) = 0 \\ k = 0, ..., N-1. \end{gathered} \tag{3.47}$$

Discrete cost function

$$\int_{kh}^{(k+1)h} C(q(t), \dot{q}(t), u(t))\, dt \approx \sum_{k=0}^{N-1} C_d(q_k, q_{k+1}, u_k) \tag{3.48}$$

Discrete Geometric constraints (path Constraint)

$$hE_1, hE_2(q_k, q_{k+1}, u_k) \geq 0 \quad k = 1, ...N-1 \tag{3.49}$$

Overall Discrete Optimal Control problem setup takes the form below.

$$
\begin{gathered}
J_d\left(q_d, u_d\right)=\sum_{k=0}^{N} C_d\left(q_k, q_{k+1}, u_k\right) \\
subject\ to \\
\left.\begin{array}{c} q_0, \dot{q}_0=0 \\ q_T, \dot{q}_T=T \end{array}\right\} Initial\ conditions \\
\left.\begin{array}{c} D_1 L_d\left(q_0, \dot{q}_0\right)+D_2 L_d\left(q_0, q_1\right)+f_0^{-}=0 \\ -D_2 L_d\left(q_N, \dot{q}_N\right)+D_1 L_d\left(q_{N-1}, q_N\right)+f_{N-1}^{+}=0 \end{array}\right\} Boundary\ conditions \\
D_1 L_d\left(q_k, q_{k+1}\right)+D_2 L_d\left(q_{k-1}, q_k\right)+f_k^{-}+f_{k-1}^{+}=0 \;\; k=1, ..., N-1\ DELE \\
hE_1, hE_2\left(q_k, q_{k+1}, u_k\right) \geq 0 \;\; k=1, ..., N-1\ Path\ constraint
\end{gathered}
\tag{3.50a}
$$

Chapter 4

METHOD SETUP FOR EXAMPLE KCM PROBLEM'S

4.1 2-link Revolute Planar Robot Problem

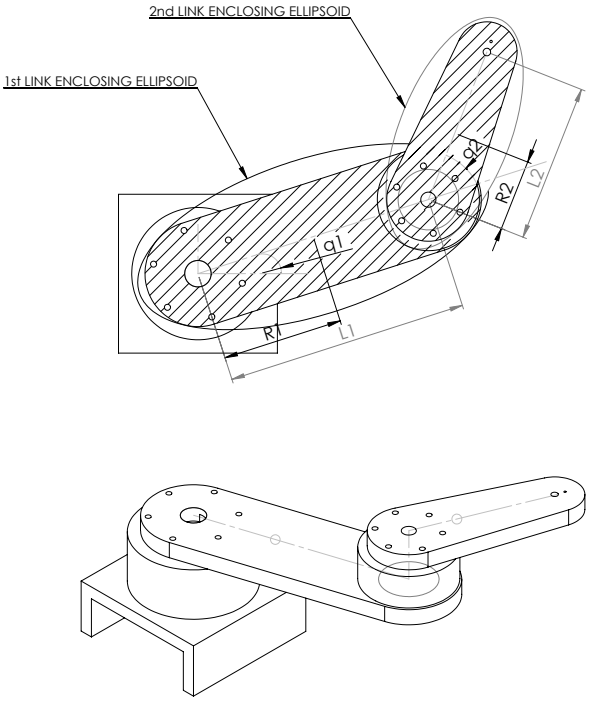

FIGURE 4.1: Schematic drawing of the experimental robot showing enclosing ellipsoids

4.1.1 Dynamics and Kinematic Equations of the Robot

The model employed is a 2-link planar manipulator with revolute joints and fully rigid links constrained at the joints. The kinematic analysis of a constrained KCM begins with the kinematics equations of its connecting serial chains. Links flexibility effects, gravity effects, viscous and coulomb friction and external noise on the torque control (assuming the system is immune to this form of error or system disturbance) are neglected. This simplified the general problem setup; however, the resulting model is still sufficient for the proposed study.
Following the derivation of the dynamic model in [71], the robots dynamic equations are derived as follows:

Position coordinates of link-1 and the corresponding Jacobian:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} r_1 \cos q_1 & 0 \\ r_1 \sin q_1 & 0 \\ 0 & 0 \end{bmatrix} \quad J_{01} = \begin{bmatrix} -r_1 \sin(q_1) & 0 \\ r_1 \cos(q_1) & 0 \\ 0 & 0 \end{bmatrix}$$

$$v_{c1} = J_{01}\dot{q}$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{z}_1 \end{bmatrix} = J_{01}\dot{q}_1 = \begin{bmatrix} -r_1 \sin(q_1)\dot{q}_1 & 0 \\ r_1 \cos(q_1)\dot{q}_1 & 0 \\ 0 & 0 \end{bmatrix}$$

and

$$||\mathrm{v}_1||^2 = {v_{c1}}^T v_{c1} = \dot{x}_1^2 + \dot{y}_1^2 + \dot{z}_1^2$$

$$T_1 = \tfrac{1}{2} m_1 ||v_1||^2 + \tfrac{1}{2} {w_1}^T I_1 w_1$$

$$T_1 = \tfrac{1}{2} m_1 \left[(-r_1 \sin(q_1)\dot{q}_1)^2 + (r_1 \cos(q_1)\dot{q}_1)^2\right] + \tfrac{1}{2} I_{z1} \dot{q}_1^2 \tag{4.1}$$

$$V_1 = m_1 g r_1 \sin q_1 \tag{4.2}$$

Position coordinates of link-2 and corresponding Jacobian:

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} l_1 \cos q_1 + r_2 \cos(q_1 + q_2) & 0 \\ l_1 \sin q_1 + r_2 \sin(q_1 + q_2) & 0 \\ 0 & 0 \end{bmatrix}$$

$$J_{12} = \begin{bmatrix} -l_1 \sin q_1 - r_2 \sin(q_1 + q_2) & -r_2 \sin(q_1 + q_2) \\ l_1 \cos q_1 + r_2 \cos(q_1 + q_2) & r_2 \cos(q_1 + q_2) \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}_2 \\ \dot{y}_2 \\ \dot{z}_2 \end{bmatrix} = J_{12}(\dot{q}_1, \dot{q}_2) = \begin{bmatrix} -l_1 \dot{q}_1 \sin q_1 - r_2 \dot{q}_1 \sin(q_1 + q_2) & -r_2 \dot{q}_2 \sin(q_1 + q_2) \\ l_1 \dot{q}_1 \cos q_1 + r_2 \dot{q}_1 \cos(q_1 + q_2) & r_2 \dot{q}_2 \cos(q_1 + q_2) \\ 0 & 0 \end{bmatrix}$$

$$v_{c2} = J_{12} \dot{q}$$

and

$$||\mathrm{v}_2||^2 = {v_{c2}}^T v_{c2} = \dot{x}_2^2 + \dot{y}_2^2 + \dot{z}_2^2$$

$$T_2 = \tfrac{1}{2} m_2 \left(\begin{array}{l} \left((-l_1 \sin q_1 - r_2 \sin(q_1 + q_2))\, \dot{q}_1 - (r_2 \sin(q_1 + q_2))\, \dot{q}_2\right)^2 + \\ \left((l_1 \cos q_1 + r_2 \cos(q_1 + q_2))\, \dot{q}_1 + (r_2 \cos(q_1 + q_2))\, \dot{q}_2\right)^2 \end{array} \right) + \tfrac{1}{2} I_{z2} (\dot{q}_1 + \dot{q}_2)^2 \tag{4.3a}$$

$$V_2 = m_2 g \left(l_1 \sin q_1 + r_2 \sin(q_1 + q_2)\right) \tag{4.3b}$$

$$L = [T_1 + V_2] - [T_1 + V_2]$$

With some calculations based on trigonometric identities, the final Lagrange equation can be reduced to give the following equation:

$$\begin{array}{c} L = \tfrac{1}{2} m_1 {r_1}^2 {q_1}^2 + \tfrac{1}{2} m_2 {l_1}^2 {q_1}^2 + \tfrac{1}{2} m_2 {r_2}^2 (\dot{q}_1 + \dot{q}_2)^2 + m_2 l_1 r_2 (\dot{q}_1^2 + \dot{q}_2 \dot{q}_1) \cos q_2 \\ -m_1 g r_1 \sin q_1 + m_2 g (l_1 \sin q_1 + r_2 \sin(q_1 + q_2) \end{array} \tag{4.4}$$

The equation of motion from the Lagrangian formulation above is derived as follows:

$$\begin{aligned}\frac{\partial L}{\partial \dot{q}_1} &= (m_1+m_2)\,{r_1}^2\dot{q}_1^2 + m_2{r_2}^2\,(\ddot{q}_1+\ddot{q}_2) + m_2 l_1 r_2\,(2\ddot{q}_1+\dot{q}_2)\cos q_2 \\ \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_1} &= (m_1+m_2)\,{r_1}^2\ddot{q}_1 + m_2{r_2}^2(\ddot{q}_1+\ddot{q}_2) + m_2 l_1 r_2\,(2\ddot{q}_1+\dot{q}_2)\cos q_2 \\ &\quad -m_2 l_1 r_2\left(2\dot{q}_1\dot{q}_2+\dot{q}_2^2\right)\sin q_2\end{aligned} \tag{4.5a}$$

$$\begin{aligned}\frac{\partial L}{\partial \dot{q}_2} &= -m_2 l_1 r_2\left(\dot{q}_1^2+\dot{q}_1\dot{q}_2\right)\sin q_2 - m_2 g r_2\cos\left(q_1+q_2\right) \\ \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_2} &= m_2{r_2}^2\left(\ddot{q}_1+\ddot{q}_2\right) + m_2 l_1 r_2\ddot{q}_1\cos q_2 - m_2 l_1 r_2\dot{q}_1\dot{q}_2\sin q_2\end{aligned} \tag{4.5b}$$

$$\begin{aligned}\tau_1 = &\left(\left(m_1{r_1}^2+m_2{l_1}^2\right)+m_2{r_2}^2+2m_2 l_1 r_2\cos q_2+I_1+I_2\right)\ddot{q}_1 + \\ &\left(m_2{r_2}^2+m_2 l_1 r_2\cos q_2+I_2\right)\ddot{q}_2 - m_2 l_1 r_2\left(\dot{q}_2^2+2\dot{q}_1\dot{q}_2\right)\sin q_2 + \\ &m_2 g r_2\cos\left(q_1+q_2\right)+\left(m_1 r_1+m_2 l_1\right)g\cos q_1\end{aligned} \tag{4.6a}$$

$$\begin{aligned}\tau_2 = &\left(m_2{r_2}^2+m_2 l_1 r_2\cos q_2\right)\ddot{q}_1+\left(m_2{r_2}^2+I_2\right)\ddot{q}_2 + \\ &m_2 l_1 r_2\dot{q}_1^2\sin q_2+m_2 g r_2\cos\left(q_1+q_2\right)\end{aligned} \tag{4.6b}$$

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = \begin{bmatrix} m_1{r_1}^2+m_2({l_1}^2+{r_2}^2+2l_1 r_2\cos q_2)+I_1+I_2 & m_2 r_2(r_2+l_1\cos q_2)+I_2 \\ m_2{r_2}^2+m_2 l_1 r_2\cos q_2+I_2 & m_2{r_2}^2+I_2\end{bmatrix} \begin{bmatrix}\ddot{q}_1\\ \ddot{q}_2\end{bmatrix} + \begin{bmatrix} -\dot{q}_2 m_2 l_1 r_2\sin q_2 & -m_2 l_1 r_2(\dot{q}_2+\dot{q}_1)\sin q_2 \\ m_2 l_1 r_2\dot{q}_1\sin q_2 & 0\end{bmatrix}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix} \tag{4.7}$$

A general state-space equation relating the dynamics of the robot can be written as:

$$\tau\left(t\right) = M(q\left(t\right))\ddot{q}\left(t\right) + C\left(q\left(t\right),\dot{q}\left(t\right)\right)\dot{q}\left(t\right) \tag{4.8}$$

With the parameters components consistent with the planar N-link robot equations in 3.8, moment about the center of mass gives the resulting generalized inertia matrix is:

$$M = \begin{bmatrix} m_1 {r_1}^2 + m_2({l_1}^2 + {r_2}^2 + 2l_1 r_2 \cos q_2) + I_1 + I_2 & m_2({r_2}^2 + l_1 r_2 \cos q_2) + I_2 \\ m_2 r_2(r_2 + l_1 \cos q_2) + I_2 & m_2 {r_2}^2 + I_2 \end{bmatrix} \tag{4.9}$$

$$C = \begin{bmatrix} -\dot{q}_2 m_2 l_1 r_2 \sin q_2 & -m_2 l_1 r_2(\dot{q}_2 + \dot{q}_1) \sin q_2 \\ m_2 l_1 r_2 \dot{q}_1 \sin q_2 & 0 \end{bmatrix} \tag{4.10}$$

q_i =joint angle, m=link mass, l_i=length of link.

r_i =Distance from previous joint to center of mass of link.

I_i =Mass Moment of inertia of links.

Parameter	**Value**	**Unit**
Length of Link 1	0.32	m
Length of Link 2	0.21	m
Mass of link 1	9.244	kg
Mass of link 2	3.529	kg
Mass moment of inertia link 1 at center of mass	0.2097	$kg.m^2$
Mass moment of inertia link 2 at center of mass	0.0206	$kg.m^2$
Link center of mass coordinate length link 1	0.16	m
Link center of mass coordinate length link 2	0.046	m

TABLE 4.1: Link parameters of experimental robot setup from [1]

4.1.2 Robot Geometric Constraints: Minimum Enclosing Ellipsoids

Using the minimum enclosing ellipsoid formulation, the tip position of the individual links 1 and 2 are defined as $p_{1_{x,y}}$ and $p_{2_{x,y}}$ respectively as follows with obstacle information derived from the robot workspace in the Cartesian plane is defined as E_1 and E_2

Ellipsoid 1	**Ellipsoid 2**
$a_1^2 = 0.065957m^2$	$a_2^2 = 0.030155m^2$
$b_1^2 = 8.8631e^{-3}m^2$	$b_2^2 = 8.8631e^{-3}m^2$
$l_{e1} = 0.16m$	$l_{e2} = 0.083545m$

TABLE 4.2: Minimum enclosing ellipsoids for 2-link robot setup

for the link 1 and 2 corresponding ellipses $[o_{d_x}, o_{d_y}]$.

$$E_1 = \frac{[(o_{d(x)}-p_{1(x)})cos\phi_1-(o_{d(y)}-p_{1(y)})sin\phi_1]^2}{{a_1}^2} + \frac{[(o_{d(y)}-p_{1(y)})cos\phi_1-(o_{d(x)}-p_{1(x)})sin\phi_1]^2}{{b_1}^2} \leq 1 \tag{4.11a}$$

$$E_2 = \frac{[(o_{d(x)}-p_{2(x)})cos\phi_1-(o_{d(y)}-p_{2(y)})sin\phi_2]^2}{{a_2}^2} + \frac{[(o_{d(y)}-p_{2(y)})cos\phi_2-(o_{d(x)}-p_{2(x)})sin\phi_2]^2}{{b_2}^2} \leq 1 \tag{4.11b}$$

4.1.3 Inactive Link Constraint Strategy (Passive Ellipsoids)

To further reduce the computation time of the problem, the algorithm is setup in a way to intelligently ignore the geometric constraints imposed on links when obstacle coordinate positions are outside its work envelope. For instance, if an obstacle is located outside the workspace of the first link $l = 1$, only successive links constraints i.e. $E_2, ..., E_N$ are active. The first link geometric constraint E_1 is therefore ignored.

4.1.4 Cost functional with Time Varying State Constraint

$$J(\tau(t), q(t)) = \frac{1}{2}\int_{t_i}^{t_f} \left\{ W_u \left(\|\tau_1(t)\|^2 + \|\tau_2(t)\|^2 \right) + W_e \|Tr(t)\|^2 \right\} dt \quad t \in [t_i, t_f] \tag{4.12}$$

4.2 Extending the Results to N-link KCM's.

4.2.1 A Redundant link KCM

To further generalize and validate the feasibility of the method setup for general N-link KCM's, the same problem set-up was extended for a manipulator with redundant link properties: A 3-link robot.

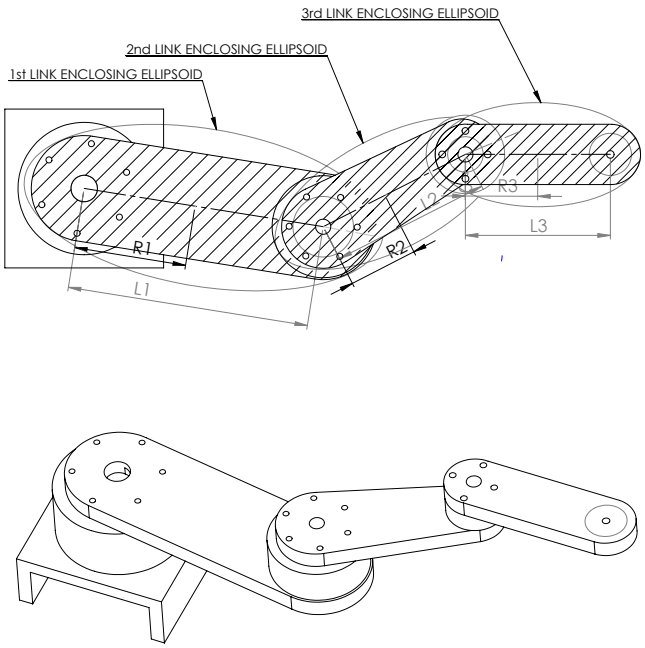

FIGURE 4.2: Schematic drawing of a redundant link robot showing enclosing ellipsoids

4.2.2 Robot Dynamics

In the same way as above, neglecting all gravity effects and assumed the links are once again none flexible, then total potential energy $v_T = 0$. Using position coordinates of link-1, 2 and 3; the respective computation of the Jacobian goes as follows:

$$T_1 = \tfrac{1}{2} m_1 ||v_1||^2 + \tfrac{1}{2} {\omega_1}^T I_1 \omega_1$$
$$T_2 = \tfrac{1}{2} m_2 ||v_2||^2 + \tfrac{1}{2} {\omega_2}^T I_2 \omega_2$$
$$T_3 = \tfrac{1}{2} m_3 ||v_3||^2 + \tfrac{1}{2} {\omega_3}^T I_3 \omega_3$$

$$T_T = \frac{1}{2} m_1 ||v_1||^2 + \frac{1}{2} {\omega_1}^T I_1 \omega_1 + \frac{1}{2} m_2 ||v_2||^2 + \frac{1}{2} {\omega_2}^T I_2 \omega_2 + \frac{1}{2} m_3 ||v_3||^2 + \frac{1}{2} {\omega_3}^T I_3 \omega_3 \quad (4.13)$$

Taking the partial derivatives of the Lagrange equation following the Euler-Lagrange formulation for a general N-link manipulator, the dynamic equations are formulated with the conventional symbol consistent as stated in equation 3.8

The dynamic equation is derived as:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} c_{12} & 0 & 0 \\ 0 & c_{22} & 0 \\ 0 & 0 & c_{33} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_2 \end{bmatrix} \tag{4.14}$$

$$\begin{gathered} c_{11} = -\left(\left(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2\right)\left(\sin q_2\left(m_2 l_1 r_2 + m_3 l_1 l_2\right) + m_3 l_1 r_3 \sin\left(q_2 + q_3\right)\right)\right) \\ -\left(\left(2\dot{q}_1\dot{q}_3 + 2\dot{q}_2\dot{q}_3 + \dot{q}_3^2\right)\left(m_3\left(l_1 r_3 \sin\left(q_2 + q_3\right) + l_2 r_3 \sin q_3\right)\right)\right) \\ c_{22} = m_2 r_2 l_1 \dot{q}_1^2 \sin q_2 + m_3 l_1 l_2 \dot{q}_1^2 \sin q_2 + m_3 r_3 l_1 \dot{q}_1^2 \sin\left(q_2 + q_3\right) \\ -m_3 r_3 l_2 \sin q_3\left(2\dot{q}_1\dot{q}_3 + 2\dot{q}_2\dot{q}_3 + \dot{q}_3^2\right) \\ c_{33} = m_3 l_1 r_3 \dot{q}_1^2 \sin\left(q_2 + q_3\right) + m_3 l_2 r_3 \dot{q}_1^2 \sin q_3 \\ +m_3 l_2 r_3 \sin q_3\left(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2\right) \end{gathered}$$

$$\begin{gathered} m_{11} = m_1 {r_1}^2 + m_2({l_1}^2 + {r_2}^2) + m_3({l_1}^2 + {l_1}^2 + {r_3}^2) + I_1 + I_2 + I_3 \\ +2(m_2 l_1 r_2 + m_3 l_1 l_2)\cos q_2 + 2m_3 l_2 r_3 \cos q_3 + 2m_3 l_1 r_3 \cos(q_3 + q_2) \\ m_{12} = m_2 {r_2}^2 + m_3({l_2}^2 + {r_2}^3) + (m_2 l_1 r_2 + m_3 l_1 l_2)\cos q_2 \\ +m_3(l_1 r_3 \cos(q_3 + q_2) + 2l_2 r_3 \cos q_3) + I_2 + I_3 \\ m_{13} = m_3({r_3}^2 + l_2 r_3 \cos q_3) + m_3 l_1 r_3 \cos(q_3 + q_2) + I_3 \\ m_{21} = m_2 {r_2}^2 + m_3({l_2}^2 + {r_2}^3) + (m_2 l_1 r_2 + m_3 l_1 l_2)\cos q_2 \\ + + m_3(l_1 r_3 \cos(q_3 + q_2) 2l_2 r_3 \cos q_3) + I_2 + I_3 \\ m_{22} = m_2 r_2^2 + m_3(l_2^2 + l_3^2) + 2m_3 l_2 r_3 \cos q_3 + I_2 + I_3 \\ m_{23} = m_3(r_3^2 + l_2 r_3 \cos q_3) + I_3 \\ m_{31} = m_3({r_3}^2 + l_1 r_3 \cos(q_3 + q_2) + l_2 r_3 \cos q_3) + I_3 \\ m_{32} = m_3(r_3^2 + l_2 r_3 \cos q_3) + I_3 \\ m_{33} = m_3 {r_3}^2 + I_3 \end{gathered}$$

4.2.3 Cost Functional with Time varying Trajectory

$$J(\tau(t), q(t)) = \frac{1}{2}\int_{t_i}^{t_f} \left\{ W_u \left(\|\tau_1(t)\|^2 + \|\tau_2(t)\|^2 + \|\tau_3(t)\|^2 \right) + W_e \|Tr(t)\|^2 \right\} dt \quad t \in [t_i, t_f] \tag{4.15}$$

4.2.4 Obstacle Avoidance /Path Constraint

Inactive ellipsoid constraints are derived in the same way as in the 2-link case. Here the robot has three minimum enclosing ellipsoids (E_1, E_2, E_3) for interacting links given as follows:

$$E_1 = \frac{[(o_{d(x)}-p_{1(x)})cos\phi_1-(o_{d(y)}-p_{1(y)})sin\phi_1]^2}{{a_1}^2} + \frac{[(o_{d(y)}-p_{1(y)})cos\phi_1-(o_{d(x)}-p_{1(x)})sin\phi_1]^2}{{b_1}^2} \leq 1 \tag{4.16a}$$

$$E_2 = \frac{[(o_{d(x)}-p_{2(x)})cos\phi_2-(o_{d(y)}-p_{2(y)})sin\phi_2]^2}{{a_2}^2} + \frac{[(o_{d(y)}-p_{2(y)})cos\phi_2-(o_{d(x)}-p_{2(x)})sin\phi_2]^2}{{b_2}^2} \leq 1 \tag{4.16b}$$

$$E_3 = \frac{[(o_{d(x)}-p_{3(x)})cos\phi_3-(o_{d(y)}-p_{3(y)})sin\phi_3]^2}{{a_3}^2} + \frac{[(o_{d(y)}-p_{3(y)})cos\phi_3-(o_{d(x)}-p_{3(x)})sin\phi_3]^2}{{b_3}^2} \leq 1 \tag{4.16c}$$

Ellipsoid 1	**Ellipsoid 2**	**Ellipsoid 3**
$a_1^2 = 0.0256m^2$	$a_2^2 = 0.0256m^2$	$a_3^2 = 0.0256m^2$
$b_1^2 = 0.0001m^2$	$b_2^2 = 0.0001m^2$	$b_3^20.0001m^2$
$l_{e1} = 0.25m$	$l_{e2} = 0.25m$	$l_{e3} = 0.25m$

TABLE 4.3: Minimum enclosing ellipsoids for 3-link robot setup

Parameter	**Value**	**Unit**
Length of Link 1	0.5	m
Length of Link 2	0.5	m
Length of Link 3	0.5	m
Mass of link 1	2.06	kg
Mass of link 2	2.06	kg
Mass of link 3	2.06	kg
Mass moment of inertia link 1 at center of mass	0.42917	$kg.m^2$
Mass moment of inertia link 2 at center of mass	0.42917	$kg.m^2$
Mass moment of inertia link 3 at center of mass	0.42917	$kg.m^2$
Link center of mass coordinate length link 1	0.25	m
Link center of mass coordinate length link 2	0.25	m
Link center of mass coordinate length link 3	0.25	m

TABLE 4.4: Link parameters of 3-link KCM robot

4.3 DCM Implementation for the 2-link and 3-link Robot Problem

The optimal control problem was setup and implemented using PSOPT (A C++ based library) as the discretization platform for the *Direct Collocation method*(DCM). The discretization scheme specified using this algorithm is global with the moving mesh to foster the capturing of the nonlinearities of the dynamics through higher concentration of nodes around the areas with higher activity to give better solution and lower error in the polynomial approximation. The resulting NLP is submitted along with the best choice of Initial guesses to a SQP algorithm (IPOPT) for computing local root solutions of the NLP. The derivatives (gradients and hessian) were computed using ADOL-C, an automatic differentiation algorithm which is motivated by the intent of helping the NLP algorithm converge mush faster. For precision scaling, the error tolerance was imposed as low as $1e^{-6}$ with scaling effects for all optimization variables to assure quality of so-called feasible solutions.

The discretization of the KCM obstacle avoidance problem is formulated using the trapezoidal and Legendre methods with automatic mesh refinement options for the time discretization intervals (nodes/collocation points). These options were chosen after a number preliminary setups were carried out for each problem scenario. The optimal solution guesses implemented were also derived in the same way.
These various problem scenarios were designed as boundary value problem with limits on the state and control variables. The settling velocity was specified to be zero at the initial and final solution time for one-shot problems. The problem was reformulated in various ways to accommodate the implementation at the experimentation phase.
The step-by-step formulation of the problem is explicitly stated in *Appendix A* and simulation results for various scenarios are given in this chapter stating the solution time and the problem information that evolved from the transcription process.

4.4 DMOC Implementation for the 2-link Robot Problem

The *Discrete Mechanics for Optimal Control*(DMOC) method was formulated in AMPL and implemented using IPOPT as the NLP solver for computing resulting local minimums of the NLP.

The Lagrange equation derived from equation 4.4 may be rewritten as follows:

$$L(q,\dot{q}) = \tfrac{1}{2}\dot{q}_1^2\left[m_1{r_1}^2 + m_2{l_1}^2 + \tfrac{1}{12}m_1{l_1}^2 + \tfrac{1}{12}m_2{l_2}^2\right] + \tfrac{1}{2}\dot{q}_2^2\left[m_2{r_2}^2 + \tfrac{1}{12}m_2{l_2}^2\right] + \dot{q}_1\dot{q}_2 m_2 l_1 {r_2}^2\cos q_2 \tag{4.17}$$

Discretizing the Lagrange equation of the system, the resulting formulation results to:

$$\delta\sum_{k=0}^{N-1} L_d(q_k, q_{k+1}) \approx \int_{k(h)}^{k+1(h)} L(q,\dot{q})dt \tag{4.18}$$

$$L(q_k, q_{k+1}, h) = \tfrac{1}{2}\left[\tfrac{q1_{k+1}-q1_k}{h}\right]^2\left[m_1{r_1}^2 + m_2{l_1}^2 + \tfrac{1}{12}m_1{l_1}^2 + \tfrac{1}{12}m_2{l_2}^2\right] + \tfrac{1}{2}\left[\tfrac{q2_{k+1}-q2_k}{h}\right]^2\left[m_2{r_2}^2 + \tfrac{1}{12}m_2{l_2}^2\right] + \left[\tfrac{q1_{k+1}-q1_k}{h}\right]\left[\tfrac{q2_{k+1}-q2_k}{h}\right]\cos\left[\tfrac{q2_{k+1}+q2_k}{2}\right]m_2 l_1 {r_2}^2 + \tfrac{1}{12}m_2{l_2}^2\left[\tfrac{q1_{k+1}-q1_k}{h}\right]\left[\tfrac{q2_{k+1}-q2_k}{h}\right] \tag{4.19}$$

$$A = \tfrac{1}{2}\left[m_1{r_1}^2 + m_2{l_1}^2 + \tfrac{1}{12}m_1{l_1}^2 + \tfrac{1}{12}m_2{l_2}^2\right]$$

$$B = \tfrac{1}{2}\left[m_2{r_2}^2 + \tfrac{1}{12}m_2{l_2}^2\right]$$

$$C = m_2 l_1 {r_2}^2$$

$$D = \tfrac{1}{12}m_2{l_2}^2$$

$$L(q_k, q_{k+1}, h) = A\left[\tfrac{q1_{k+1}-q1_k}{h}\right]^2 + B\left[\tfrac{q2_{k+1}-q2_k}{h}\right]^2 + C\left[\tfrac{q1_{k+1}-q1_k}{h}\right]\left[\tfrac{q2_{k+1}-q2_k}{h}\right]\cos\left[\tfrac{q2_{k+1}+q2_k}{2}\right] + D\left[\tfrac{q1_{k+1}-q1_k}{h}\right]\left[\tfrac{q2_{k+1}-q2_k}{h}\right] \tag{4.20}$$

The intermediate control forces (Torque at each joint) at the discretized points (grid) moving the system from the corresponding initial state to the end point $q_i = q_1, q_2, q_3, ..., q_N$ is specified as follows for :

$$F_L(\tau_1, \tau_2) = \begin{bmatrix} \tau_1 - \tau_2 \\ \\ \tau_2 \end{bmatrix}$$

Taking two set of discrete Lagrange control forces and its coupled sum for balancing of force dissipation;

$$f_d^+ \left(q_k, q_{k+1}, u_k, h\right) = \int_{hk}^{h(k+1)} f_L \left(q, \dot{q}, \tau\right) \cdot \frac{\partial q}{\partial q_{k+1}} dt \tag{4.21a}$$

$$f_d^- \left(q_k, q_{k+1}, u_k, h\right) = \int_{hk}^{h(k+1)} f_L \left(q, \dot{q}, \tau\right) \cdot \frac{\partial q}{\partial q_k} dt \tag{4.21b}$$

A Midpoint Rule is used to compute the state variables given as:-$\frac{q_{k+1}+q_k}{2}$, $\frac{q_{k+1}-q_k}{h}$ to approximate the integrals and the time stepping integrator for a balance between accuracy and computation speed. The time step used for this problem is given as $h = 2\text{x}10^{-3}$.

Taking partial derivatives at each discrete time step, an equivalent to the equation of motion of the system derived as follows:

$$\begin{gathered} \frac{\delta}{\delta q_k} \ (Partial\ derivatives\ wrt\ "k") \\ \frac{q_{k+1}-q_k}{h} = -\frac{1}{h}, \left[\frac{q_{k+1}-q_k}{h}\right]^2 = -\frac{2}{h}\left[\frac{q_{k+1}-q_k}{h}\right] \\ \frac{q_k+q_{k+1}}{2} = \frac{1}{2}, \left[\frac{q_k+q_{k+1}}{2}\right]^2 = \left[\frac{q_k+q_{k+1}}{2}\right] \end{gathered} \tag{4.22}$$

$$\begin{gathered} \frac{\delta}{\delta q_{k+1}} (Partial\ derivatives\ wrt\ "k+1") \\ \frac{q_{k+1}-q_k}{h} = \frac{1}{h}, \left[\frac{q_{k+1}-q_k}{h}\right]^2 = \frac{2}{h}\left[\frac{q_{k+1}-q_k}{h}\right] \approx \frac{2}{h}\left[\frac{q_k-q_{k-1}}{2}\right] \\ \frac{q_k+q_{k+1}}{2} = \frac{1}{2}, \left[\frac{q_k+q_{k+1}}{2}\right]^2 = \left[\frac{q_k+q_{k+1}}{2}\right] \approx \left[\frac{q_{k-1}+q_k}{2}\right] \end{gathered}$$

$$D_1L_d(q_k,q_{k+1}) = -\tfrac{2}{h}\left[\tfrac{q1_{k+1}-q1_k}{h}\right]A - \tfrac{2}{h}\left[\tfrac{q2_{k+1}-q2_k}{h}\right]B + D\left[-\tfrac{1}{h}\tfrac{q2_{k+1}-q2_k}{h} - \tfrac{1}{h}\tfrac{q1_{k+1}-q1_k}{h}\right] + C\begin{bmatrix} -\left[\tfrac{1}{h}\tfrac{q2_{k+1}-q2_k}{h}\cos\tfrac{q2_{k+1}+q2_k}{2}\right] - \left[\tfrac{1}{h}\tfrac{q1_{k+1}-q1_k}{h}\cos\tfrac{q2_{k+1}+q2_k}{2}\right] \\ -\left[\tfrac{1}{2}\tfrac{q1_{k+1}-q1_k}{h}\tfrac{q2_{k+1}-q2_k}{h}\sin\tfrac{q2_{k+1}+q2_k}{2}\right] \end{bmatrix} \tag{4.23a}$$

$$D_2L_d(q_{k-1},q_k) = \tfrac{2}{h}\left[\tfrac{q1_k-q1_{k-1}}{h}\right]A + \tfrac{2}{h}\left[\tfrac{q2_k-q2_{k-1}}{h}\right]B + D\left[\tfrac{1}{h}\tfrac{q2_{k+1}-q2_k}{h} + \tfrac{1}{h}\tfrac{q1_{k+1}-q1_k}{h}\right] + C\begin{bmatrix} \left[\tfrac{1}{h}\tfrac{q2_{k+1}-q2_k}{h}\cos\tfrac{q2_{k+1}+q2_k}{2}\right] + \left[\tfrac{1}{h}\tfrac{q1_{k+1}-q1_k}{h}\cos\tfrac{q2_{k+1}+q2_k}{2}\right] \\ -\left[\tfrac{1}{2}\tfrac{q1_{k+1}-q1_k}{h}\tfrac{q2_{k+1}-q2_k}{h}\sin\tfrac{q2_{k+1}+q2_k}{2}\right] \end{bmatrix} \tag{4.23b}$$

$$\begin{aligned} f_d^-(q_k,q_{k+1},\tau_k) &= \tfrac{h}{2}f\left[\tfrac{q_{k+1}+q_k}{2},\tfrac{q_{k+1}-q_k}{h},\tau_k+1/2\right] \\ f_d^+(q_k,q_{k+1},\tau_k) &= \tfrac{h}{2}f\left[\tfrac{q_{k+1}+q_k}{2},\tfrac{q_{k+1}+q_k}{h},\tau_k+1/2\right] \\ f_d^- &= f_d^+ = \tfrac{h}{4}\left[f_k + f_{k+1}\right] \\ \tau_k+1/2 &= \tau\left[\tfrac{t_k+t_{k+1}}{2}\right] \end{aligned} \tag{4.24a}$$

Discrete Euler-Lagrange Equation (Equation of motion)

$$\begin{gathered} D_1L_d\left(q_k,q_{k+1}\right) + D_2L_d\left(q_{k-1},q_k\right) + f_d^-\left(q_k,q_{k+1},\tau_k\right) + f_d^+\left(q_k,q_{k+1},\tau_{k-1}\right) = 0 \\ k = 0,...,N-1. \end{gathered} \tag{4.25}$$

Discrete cost function

The discrete equivalent of the performance index in equation is:

$$J_d(q_d,\tau_d) = \frac{1}{2}\sum_{k=0}^{N-1} W_u\|\tau_1\|^2 + W_u\|\tau_2\|^2 + W_e\left\|\left[\frac{qr_{k+1}-qr_k}{h}\right] - \left[\frac{qr_{k+1}+qr_k}{2}\right]\right\|^2 \tag{4.26}$$

Discrete Geometric constraints (Path Constraint)

The minimum enclosing geometric constraint joint parameters are also discretized as follows:

$$\begin{bmatrix} p_1(x) \\ p_1(y) \end{bmatrix} = l_{e1}\begin{bmatrix} \cos(q2_k) \\ \sin(q2_k) \end{bmatrix}$$

$$\begin{bmatrix} p_2(x) \\ p_2(y) \end{bmatrix} = l_1 \begin{bmatrix} \cos(q1_k) \\ \sin(q2_k) \end{bmatrix} + l_{e2} \begin{bmatrix} \cos(q1_k + q2_k) \\ \sin(q1_k + q2_k) \end{bmatrix}$$

Overall discrete optimal control problem setup takes the form:

$$\min_{\tau} J_d(q_d, \tau_d)$$

$$subject\ to$$

$$\left.\begin{aligned} q_0, \dot{q}_0 = 0 \\ q_T, \dot{q}_T = T \end{aligned}\right\} \text{Initial Conditions}$$

$$\left.\begin{aligned} D_1 L_d(q_0, \dot{q}_0) + D_2 L_d(q_0, q_1) + f_0^- = 0 \\ D_1 L_d(q_N, \dot{q}_N) - D_2 L_d(q_{N-1}, q_{N-1}) + f_{N-1}^+ = 0 \end{aligned}\right\} \text{Boundary Conditions}$$

$$D_1 L_d(q_k, q_{k+1}) + D_2 L_d(q_{k-1}, q_k) + f_k^- + f_{k-1}^+ = 0 \quad k = 1, ...N-1 \text{ DELE}$$

$$hE_1, hE_2(q_k, q_{k+1}, u_k) \geq 0 \quad k = 1, ..., N-1 \text{ Path Constraint}$$

Chapter 5

SIMULATIONS AND EXPERIMENTAL RESULTS

5.1 Simulation Results

The simulations were carried out with indiscriminately chosen problem type scenarios with respect to the robot initial and final desired configuration and obstacle locations. The motion problem investigated are point-to-point and predefined trajectory tracking control designed with different mixes of possible events for cases with obstacles located within and outside the robot work envelope.
Limits are imposed on the state and control variables to put practicality into consideration and various logical boundary conditions (such as zero velocity at the initial and/or final time) were incorporated to test the performance of the methods. The problem was also designed in general as fixed terminal state or free terminal state and as fixed terminal-time OCP's.

Various options for the discretization algorithm were explored for fine tuning of the results and the forgoing code is stated in the appendix (DCM and DMOC). The initial guess was intuitively specified for the NLP solver to work with and this choice has no full proof direction other than the user's prior knowledge of the likely solution. Data was collected and the quantities of paramount interest to the optimization process are as follows:

- The speed at which the problem was solved.
- The path following accuracy of the trajectory where a reference trajectory was specified.
- Accuracy i.e Cost function evaluation
- Number of iterations used to solve a problem.
- Number of inequality, equality constraints generated during discretization.
- Relative local errors.

The problem setup was implemented using C++ on an INTEL® based PC with a Dual-Core processor. UBUNTU 11.04®; A Linus Based Kernel was the operating system with 2GB RAM and 2.67GHz processor speed platform. Discretization was done for the collocation method and DMOC using PSOPT and a self-designed algorithm respectively. IPOPT® was used as the sequential quadratic programming algorithm to solve resulting NLP's for both methods.

5.1.1 Problem Scenario-1

A 2-link KCM robot problem with specified reference trajectory is setup as an OCP with free terminal state and fixed terminal-time t_f with state constraints in form of a time varying trajectory profile to command the motion along a predefined path.
A time varying function defining the desired trajectory for the problem set is defined as a straight line trajectory with a smooth velocity profile as depicted below. This trajectory is used to assess the performance of the algorithm when the system is further constrained with a time-varying state constraint as such as the smooth sinusoidal velocity profile shown below.

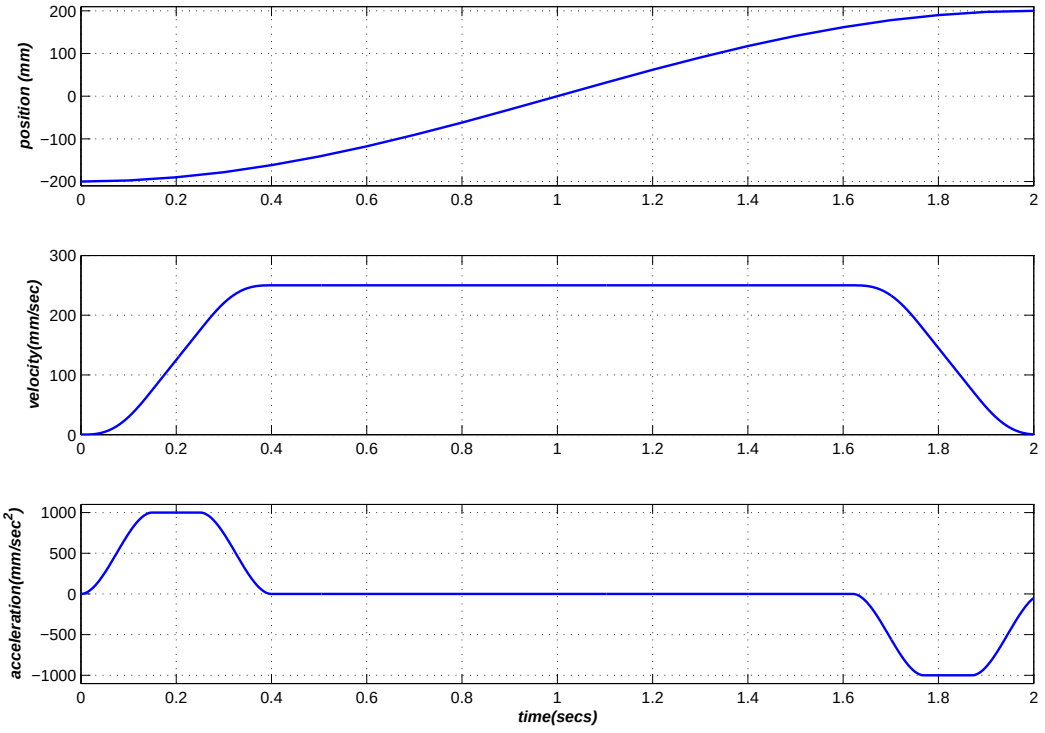

FIGURE 5.1: A 1-DOF smooth velocity profile used to command the robot states for time 0-2 seconds.

At the initial run time, an obstacle is located in the robot workspace coordinate at x=0.43, y=0.0 and the initial robot joint pose (θ_1,θ_2) is given as as: (0,π/3). Time discretization interval (nodes) = 12 for the discretization process.

CPU time (seconds)	2.2e-01
Optimal (unscaled) cost function value	4.435694e-02
Phase Initial time(seconds)	0
Phase final time(seconds)	2
Number of Iterations	21
Phase maximum relative local error	1.374592e-02
Number of objective function evaluations	51
Number of objective gradient/constraint Jacobian evaluations	22
Number of equality/inequality constraint evaluations	40
Number of Lagrangian Hessian evaluations	0

TABLE 5.1: Problem scenario-1: Optimization solution summary report.

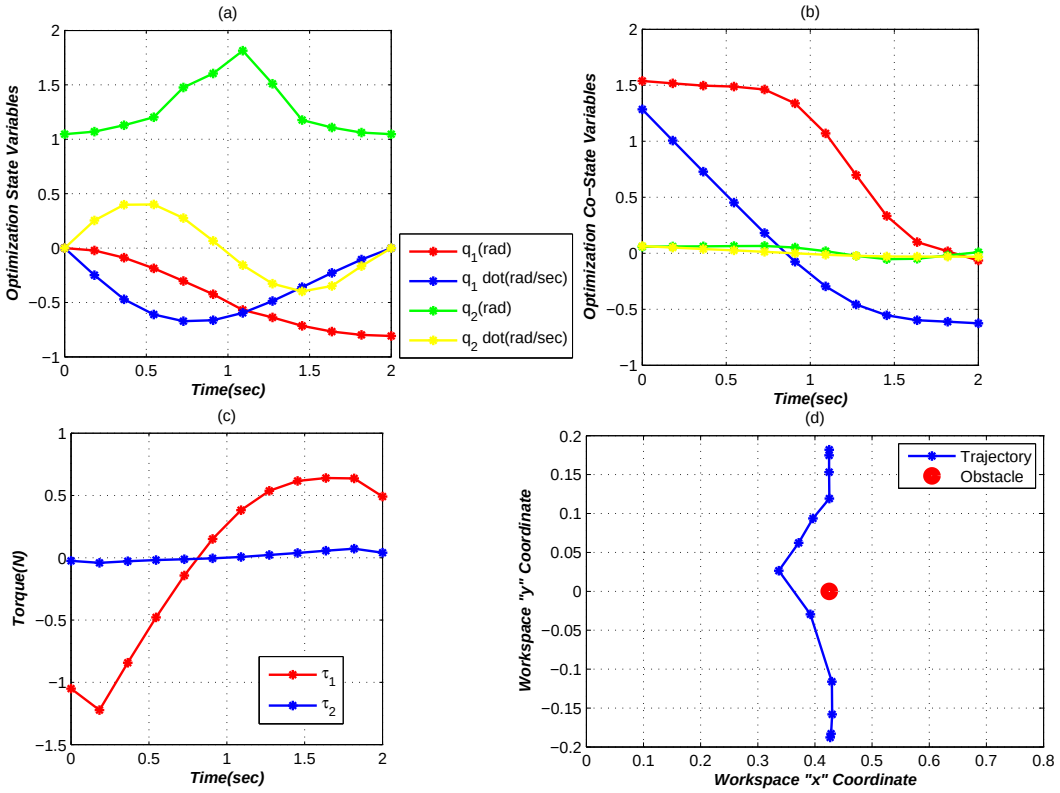

FIGURE 5.2: Optimization output plots (Problem scenario-1): (a) The optimal state variables, (b) The Co-state variables, (c) The optimal controls (torque) with respect to time and (d) The optimal trajectory in the robot workspace coordinate.

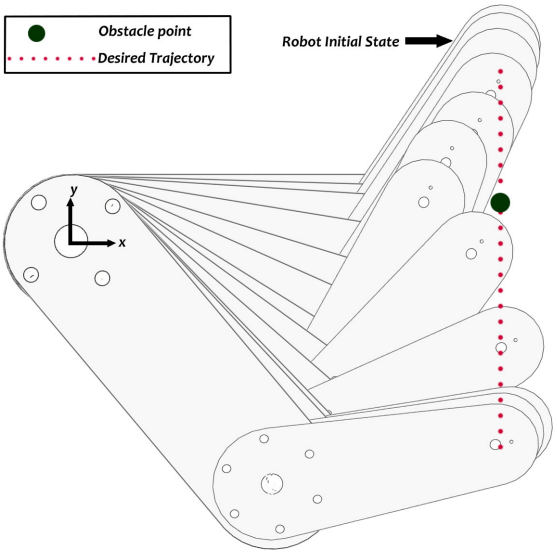

FIGURE 5.3: Robot motion profile showing the link configuration at equal time shots from the generated optimal trajectory (Problem scenario-1).

5.1.2 Problem Scenario-2

The problem setup in 5.1.1 but implemented with different choice discretization points of $N = 20$ and different initial guess.

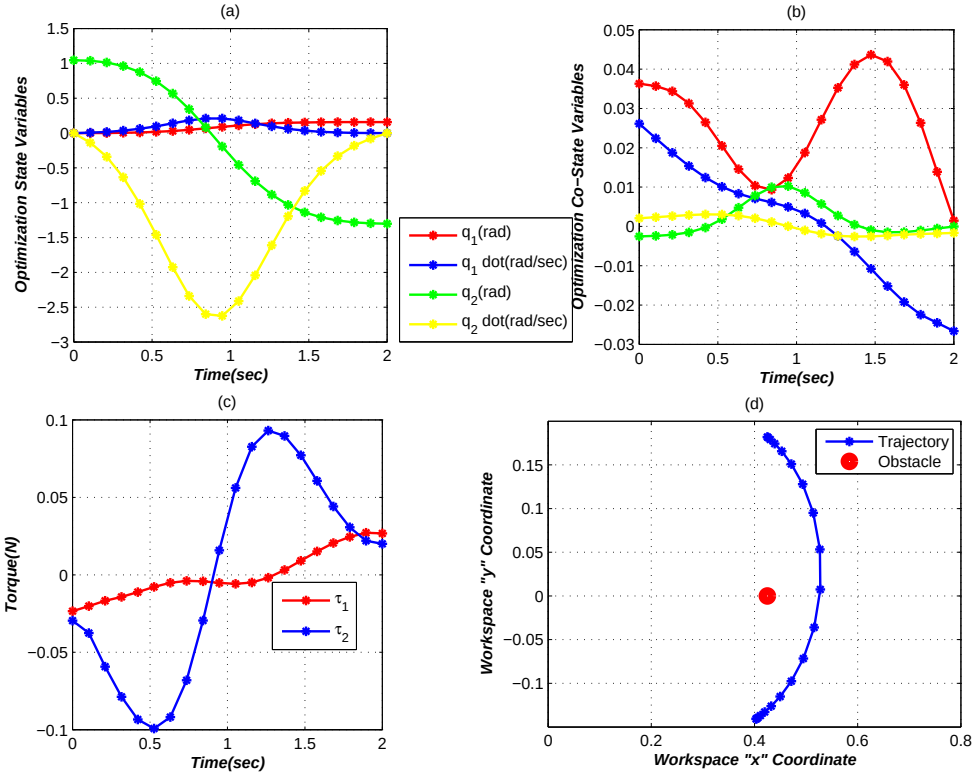

FIGURE 5.4: Optimization output plots (Problem scenario-2): (a) The optimal state variables, (b) The Co-state variables, (c) The optimal controls (torque) with respect to time and (d) The optimal trajectory in the robot workspace coordinate.

CPU time (seconds)	1e-01
Optimal (unscaled) cost function value	5.7322e-03
Phase Initial time(seconds)	0
Phase final time(seconds)	2
Number of Iterations	42
Phase maximum relative local error	2.8135e-04
Number of objective function evaluations	46
Number of objective gradient/constraint Jacobian evaluations	43
Number of equality /inequality constraint evaluations	46
Number of Lagrangian Hessian evaluations	0

TABLE 5.2: Problem scenario-2: Optimization solution summary report.

5.1.3 Problem Scenario-3

The problem setup in 5.1.1 but different implemented with different discretization points of $N = 12$ and different initial guess as well as different obstacle coordinate as x=-0.30, y=-0.20.

CPU time (seconds)	2.9e-01
Optimal (unscaled) cost function value	8.085193e-01
Phase Initial time(seconds)	0
Phase final time(seconds)	2
Number of Iterations	42
Phase maximum relative local error	2.278680e-2
Number of objective function evaluations	51
Number of objective gradient/ constraint Jacobian evaluations	46
Number of equality/inequality constraint evaluations	40
Number of Lagrangian Hessian evaluations	0

TABLE 5.3: Problem scenario-3: Optimization solution summary report.

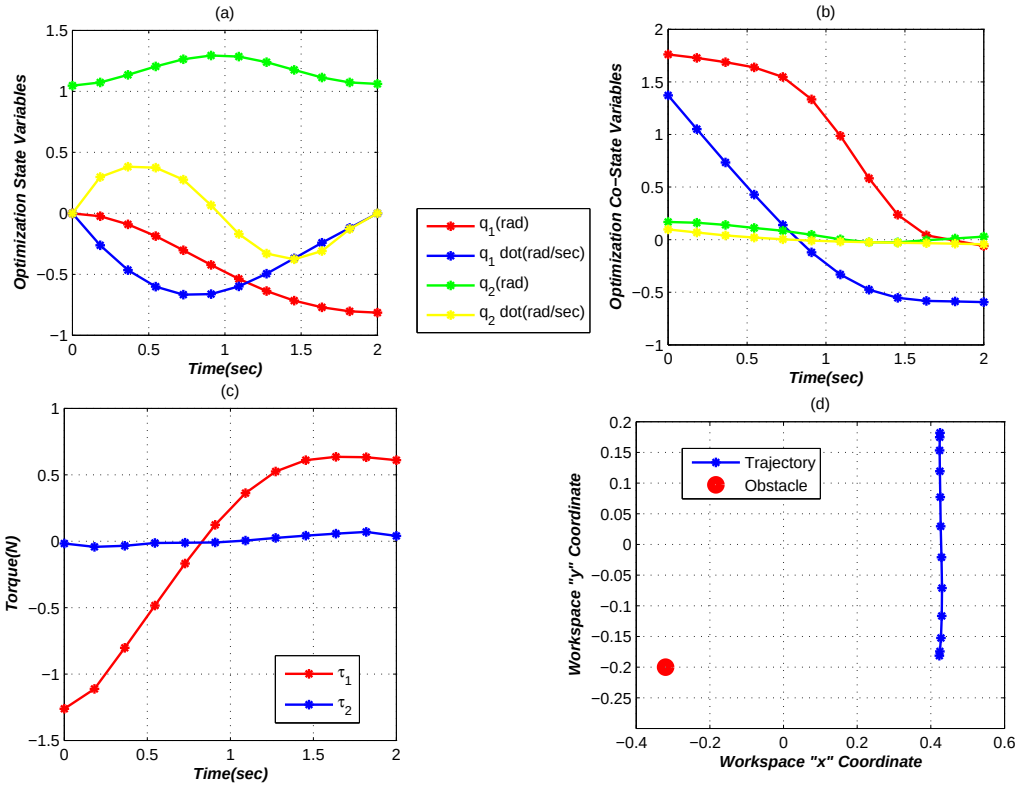

FIGURE 5.5: Optimization output plots (Problem scenario-3): (a) The optimal state variables, (b) The Co-state variables, (c) The optimal controls (torque) with respect to time and (d) The optimal trajectory in the robot workspace coordinate.

5.1.4 Problem Scenario-4

Point-to-point motion control specified as a fixed terminal-state q_t and fixed terminal-time (t_f) OCP. Obstacle location at runtime is specified as x=0.30, y=0.25 coordinate solved with 24 discretization points. Initial and desired final position of the manipulator joints θ_1 and θ_2 is chosen as: $(0,\pi/6)$ and $(-\pi/4,\pi/4)$ respectively.

CPU time (seconds)	3.8e-01
Optimal (unscaled) cost function value	2.676998e+00
Phase Initial time(seconds)	0
Phase final time(seconds)	5
Number of Iterations	64
Phase maximum relative local error	3.006598e-03
Number of objective function evaluations	65
Number of objective gradient/constraint Jacobian evaluations	65
Number of equality/inequality constraint evaluations	128
Number of Lagrangian Hessian evaluations	0

TABLE 5.4: Problem scenario-4: Optimization solution summary report.

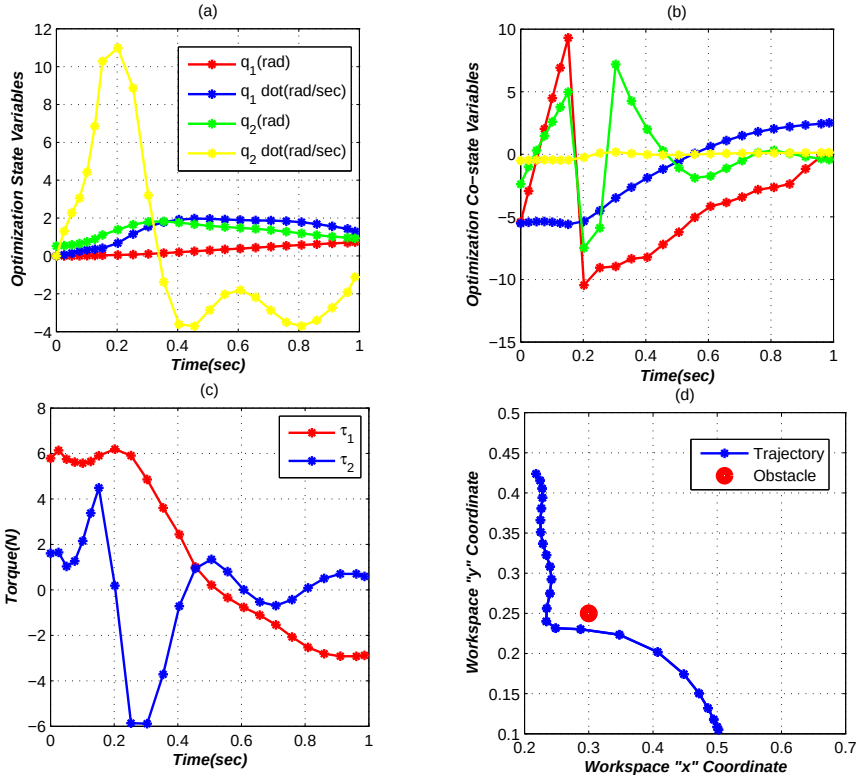

FIGURE 5.6: Optimization output plots (Problem scenario-4): (a) The optimal state variables, (b) The Co-state variables, (c) The optimal controls (torque) with respect to time and (d) The optimal trajectory in the robot workspace coordinate.

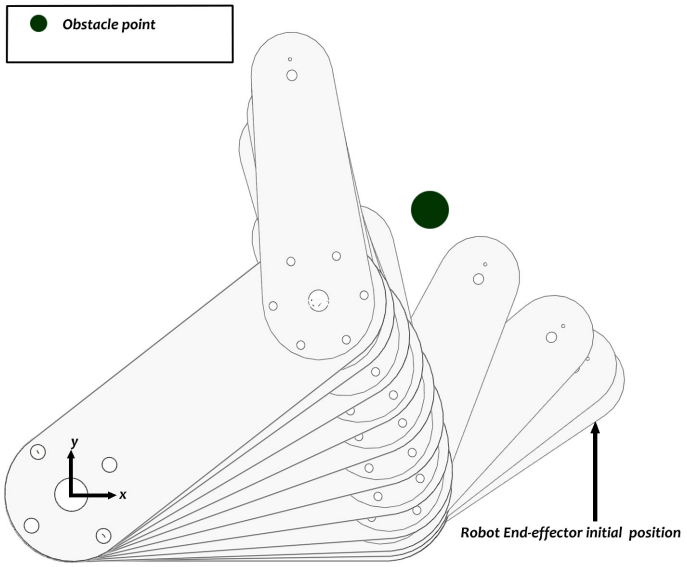

FIGURE 5.7: Robot motion profile showing the link configuration at equal time shots from the generated optimal trajectory (Problem scenario-4).

5.1.5 Problem Scenario-5

Point-to-Point motion control modeled as a fixed terminal-state and fixed end-time OCP. Obstacle location at initial runtime is x=0.30, y=0.20 coordinate solved with 20 discretization points. Initial and final position for the joints θ_1 and θ_2 is given as: (0,π/3) and (-π/4,π/4).

CPU time (seconds)	4.6e-02
Optimal (unscaled) cost function value	2.1436e-03
Phase Initial time(seconds)	0
Phase final time(seconds)	2
Number of Iterations	9
Phase maximum relative local error	1.135067e-02
Number of objective function evaluations	25
Number of objective gradient/constraint Jacobian evaluations	14
Number of equality/inequality constraint evaluations	14
Number of Lagrangian Hessian evaluations	0

TABLE 5.5: Problem scenario-5: Optimization solution summary report.

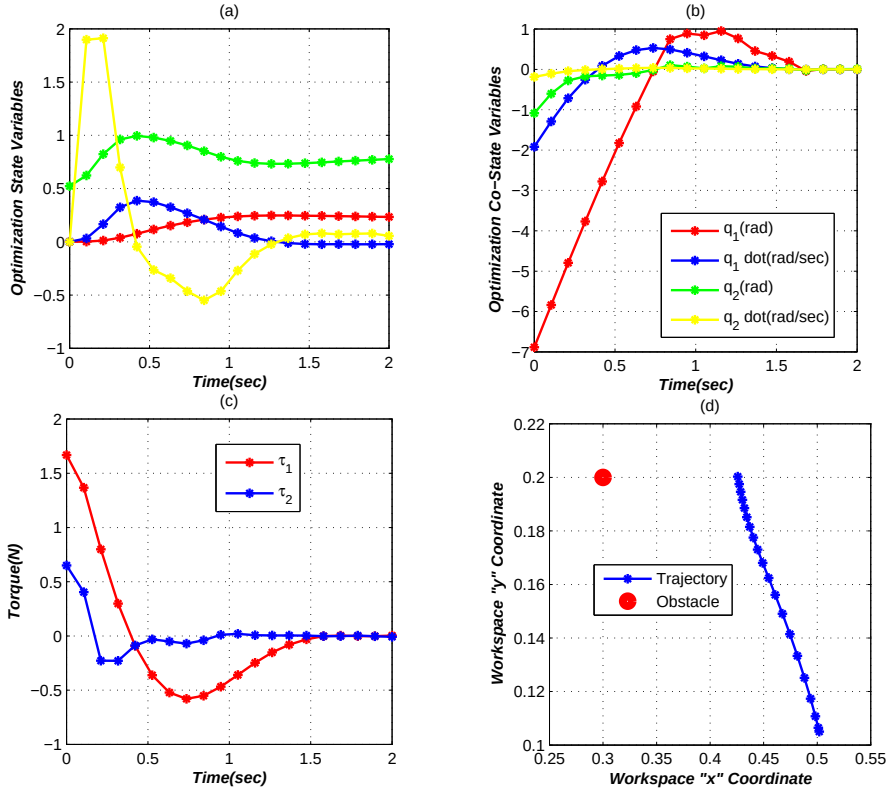

FIGURE 5.8: Optimization output plots (Problem scenario-5): (a) The optimal state variables, (b) The Co-state variables, (c) The optimal controls (torque) with respect to time and (d) The optimal trajectory in the robot workspace coordinate.

5.1.6 Problem Scenario-6

3 link KCM problem with specified reference Trajectory similar to the problem setup in scenario 1 and modeled as a fixed terminal time (T_f) and free terminal state OCP. Obstacle located at x=0.96, y=0.25 workspace coordinate and solved with 12 discretization points.

CPU time (seconds)	8.7e-01
Optimal (unscaled) cost function value	7.569988e01
Phase Initial time(seconds)	0
Phase final time(seconds)	1.8
Number of Iterations	71
Phase maximum relative local error	1.585971e-02
Number of objective function evaluations	102
Number of objective gradient/constraint Jacobian evaluations	72
Number of equality/inequality constraint evaluations	102
Number of Lagrangian Hessian evaluations	0

TABLE 5.6: Problem scenario-6: Optimization solution summary report.

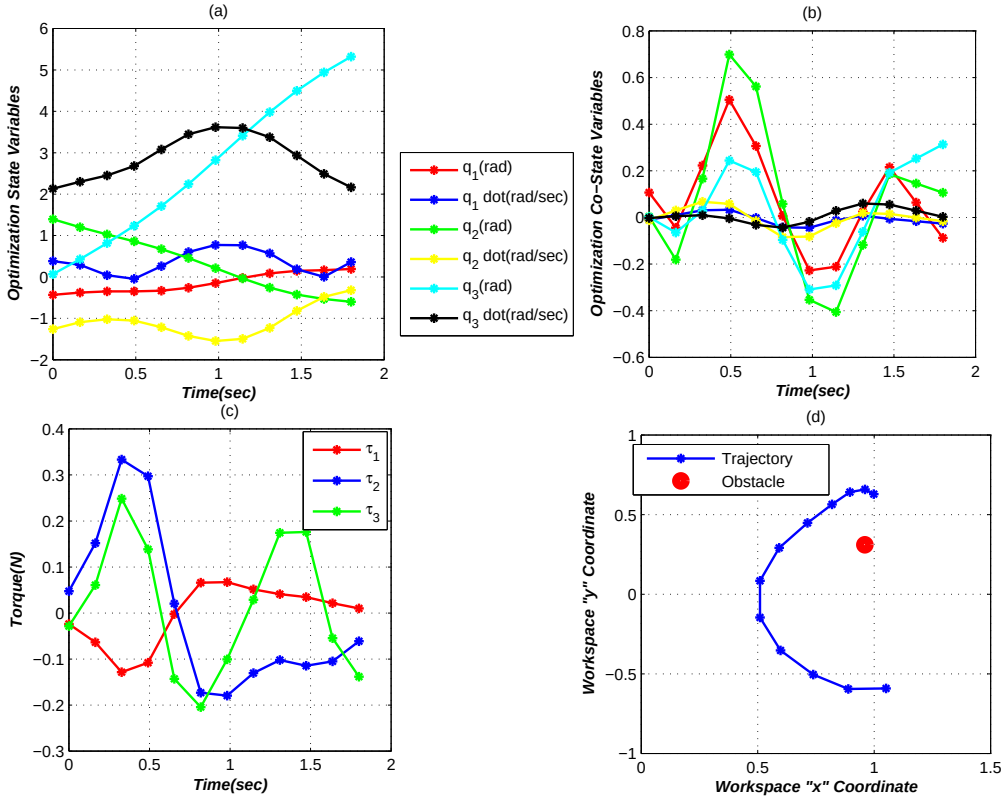

FIGURE 5.9: Optimization output plots (Problem scenario-6): (a) The optimal state variables, (b) The Co-state variables, (c) The optimal controls (torque) with respect to time and (d) The optimal trajectory in the robot workspace coordinate.

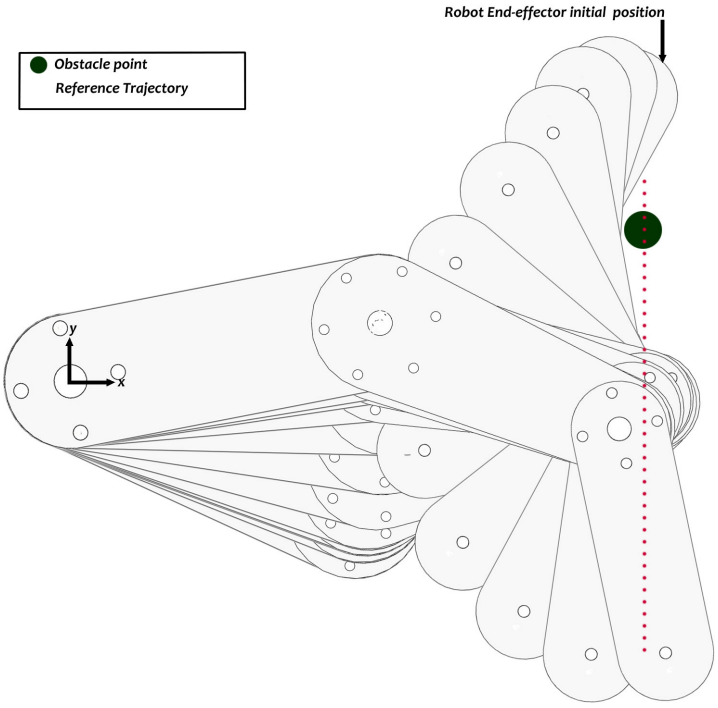

FIGURE 5.10: Robot motion profile showing the link configuration at equal time shots from the generated optimal trajectory (Problem scenario-6).

5.2 Comparison Between the Solutions from the DMOC and DCM Algorithm

This section compares both methods in terms accuracy and speed. Based on the solution time, the objective function value, the size/dimension of the NLP resulting from the discretization scheme with respect to the added equality constraint and Jacobian constraint evaluation. The following deductions are made for variants of the problem setup in simulation Scenario 1.

Discretization Intervals (Nodes)	DMOC		Direct Collocation Method	
	Obj_function value xE^{-5}	Solution time(sec)	Obj_function value xE^{-3}	Solution time(sec)
12	8.88	0.18	3.49	0.22
14	8.78	0.21	3.34	0.33
16	8.52	0.34	3.26	0.46
18	8.32	0.37	3.19	0.78
20	8.30	0.35	3.07	0.89
24	8.30	0.47	3.03	1.03
30	8.30	1.20	3.03	1.10

FIGURE 5.11: Optimization summary report of DMOC VS Collocation Method for 2-link Robot Problem with robot dynamics as only constraint.

Discretization Intervals (Nodes)	DMOC Method				Direct Collocation Method			
	Obj_function value	Solution time(sec)	Number of Iterations	Constraint Jacobian	Obj_function value	Solution time(sec)	Number of Iterations	Constraint Jacobian
12	1.783-03	0.18	12	13	4.435e-02	0.22	21	22
20	3.105e-03	0.21	22	23	5.733e-02	0.33	36	37
24	1.406e-03	0.34	34	35	6.033e-2	0.46	39	40
30	3.005e-03	0.37	34	35	7.451e-02	0.778	42	43
35	2.969e-03	0.35	34	35	1.238e-2	0.89	47	48
40	3.381e-03	0.47	53	54	2.504e-02	1.03	52	53
45	3.346e-03	1.20	310 (Ill conditioned solution)	313	2.504e-02	1.1	53	54

FIGURE 5.12: Optimization summary report of DMOC VS Collocation Method for 2-link Robot problem with obstacle avoidance and time varying trajectory as constraints.

From the tables above, the following deductions about the methods could be inferred:

1. The DMOC method shows lesser iterations in computing solutions compared to the DMOC of the resulting NLP's after discretization. For the same number of discretization steps, the resulting inequality constraint generated from the discretization process was less in the DMOC setup compared to the DCM showing a smaller size NLP accordingly. This accounts for the difference in iteration steps used for computing solutions in each case. The computation for deriving the solutions was also consistent with the iteration as it took longer to solve for NLP's discretized using the DCM compared to the DMOC setup.

2. The accuracy of the solution for both methods greatly depends on the choice of discretization steps. The DMOC method showed less sensitivity to variations in the number of discretization steps compared to the collocation method.
   The DMOC solutions experienced ill-conditioning only at points where the number of discrete time steps was less than or exceeded certain convergence bounds. The collocation method was noticed to have ill-conditioned solutions and a high variance in the solution when the number of nodes was not suitable for particular quadrature rules in computing the integral terms of the cost. Ill condition was also observed when the higher polynomials approximating the ODE representing the systems dynamics were not conditioned suitably for the number of collocation points for the DCM. This seemed to be prevalent in cases where discretization was done locally.
   The moving mesh method (global discretization) and more suitable quadrature rules for better approximation helped resolved these optimization issues.
   The accuracy of the DCM solutions may also be bettered by employing the appropriate time stepping integrator where a midpoint rule is not most efficient. The collocation method is superior in this sense because of its global discretizaton process which captures the nonlinearities in the dynamics more accurately.

3. The DMOC shows less sensitivity to the resulting solution when the nodes are changed unlike the DCM. This makes the setup easier to use in terms of choice and nodal parameters especially where there is little information about the system in question. In both methods, ill-conditioned solutions were observed when discretization interval did not fit the problem structure in keeping the variation integrator symplectic during the transcription process.

4. In both methods, the optimization parameters are both the control and state variables which is prominently reflected in the good handling of limits on the states and controls.

5. The DMOC setup uses the forward/backward forcing ratios to represent the force dissipation which also in essence helps the momentum/energy conservation and reduces ill conditioning which may arise based on this control related constraints.

6. The problem setup for the DMOC is very straight forward and much easier to formulate compared to the collocation method because the discretization was done directly in the configuration space.

5.3 Experimental Setup

5.3.1 Hardware and Software

A 2-link robot arm (planar configuration) is the experimental test bed used to verify the simulation results. Parameters are as stated in table 4.1. Obstacle information in the workspace and information from the encoders at the robots joints specifying the robots present state/pose are sent to the optimal control desktop computing the optimal trajectory using UDP data packets established by a stable bidirectional communication between the real-time computer running the servo-controller and a computer processor running the optimal control solvers. In the same way, the optimal trajectories are fed back as input commands for the servo-controller from the optimal control solver machine

The task may be summarized as follows:

- The original problem is setup in the optimal control solver PC which computes the set of optimal state trajectory $q(t)^*$. This is implemented as a continuously running loop which is triggered on and off by a control flag in the form of UDP data packets carrying information about the robot workspace specifying the initial and final optimization state variables and the prediction and control horizon for the problem.

- The robot real-time controller sends the trigger and the necessary information needed by the solver computer such as: the most recent obstacle locations at the solver run-time, present link configurations and the desired terminal states of the robot.

- On triggering, the algorithm checks to confirm the variance in the robot present state and its desired state, and based on this case structure, computes collision-free optimal trajectories to move the robot to its desired final state. It computes this fractional optimal trajectory and sends inputs (optimal trajectory) to the robot real-time controller.

- Based on the current information of the robot environment variables (new obstacle position) at the time the solution arrives, the controller computes torque commands moving the robot along the safe point commanded considering the obstacle coordinate locations at the arrival time of the solution.
- The entire process is done in successive time steps until the robot reaches its final desired location.

The hardware components comprise of the following:

- 2-LINK planar manipulator Robot powered by servo drive motors at the end of each link dictating revolute joint motion as show in the figure below.
- LINUX KERNEL UBUNTU 11.04® DESKTOP COMPUTER. 2GB RAM processor, 1GB Network card with Dual-Core processor. This runs the optimization softwares: IPOPT®/PSOPT®/ADOLC ®/AMPL®.
- COMPACTRIO 9022® REALTIME COMPUTER: The real-time computer running the servo controller is designed by National Instrument® with LABVIEW2011© as the development environment for the controller design. The machine is equipped with a 2GB processor with FPGA modules to interface the robot components.The Ethernet port has a LAN data transfer speed of 1GB.
- Display mode to monitor Robot configuration during motion control.

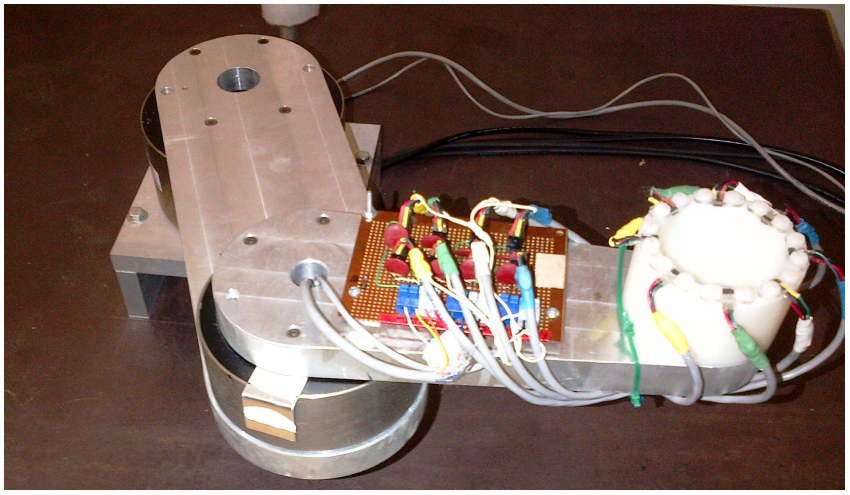

FIGURE 5.13: 2-Link KCM robot(Planar configuration).

5.3.2 Bidirectional Communication

There are quite a number of ways to communicate between computers. For this application, the most paramount properties for the choice of the communication link were the following:

- Fastest rate of data transfer for the initiative of implementing online real-time solutions updates.
- The communication link should displace the smallest amount of processor speed in the individual computers.
- Reliability of the communication to ensure the data sent across the network is correct with correctly ordered messages The data sent to the recipient arrives in the same order in which they were sent.
- The network structure is stable and bidirectional linkup/binding is fast enough for real-time control.
- Memory congestion control: No buffer/memory is permanently used up during the data transfer process.

The conventional communication setup in commercially made computers is the Internet/Intranet Protocols (IP) and the communication modem platform is usually via:

1. Serial Communication Link Ports (COM PORTS)
2. Wireless network ports
3. USB ports.
4. Ethernet networks

The UDP internet protocol transfer technology has two major problems. The possible poor ordering of data at arrival and the unreliability of the data sent across a network. The ordering problem is solved by sending the UDP datagram in one shot as a datagram at each solution iteration point. For example, with a discretization steps as 40, corresponding sample control actions for the 2-link KCM results in 80 point sets solution commands. All the data points are sent as one datagram and a clever measure is used to

divide the data set on arrival at the real-time controller. This ensures and solves the over flogged issue of possible UDP unreliability. With an exceptionally high discretization set at 100 points, the max size of datagram is still a lot less than the maximum byte which could be sent as prescribed by standard network data transfer.

The problem of unreliability in data ordering is checked by a simple flag trigger. The real-time machine waits on a flag which triggers the controller to start-up at successive stages after motion commands have been generate by the optimal control solver PC. This flag checking is also repeatedly done before any data from the solver computer can be trusted for use. This also ensures no clog in memory because of the data transfers. In the same way, the triggering action is used to automatically initialize the trajectory generation process in the solver computer to safely accommodate changing robot environment variables. Several test-runs were carried out to verify the data sent across the network were well ordered correctly and efficiently with respect to buffer memory usage.

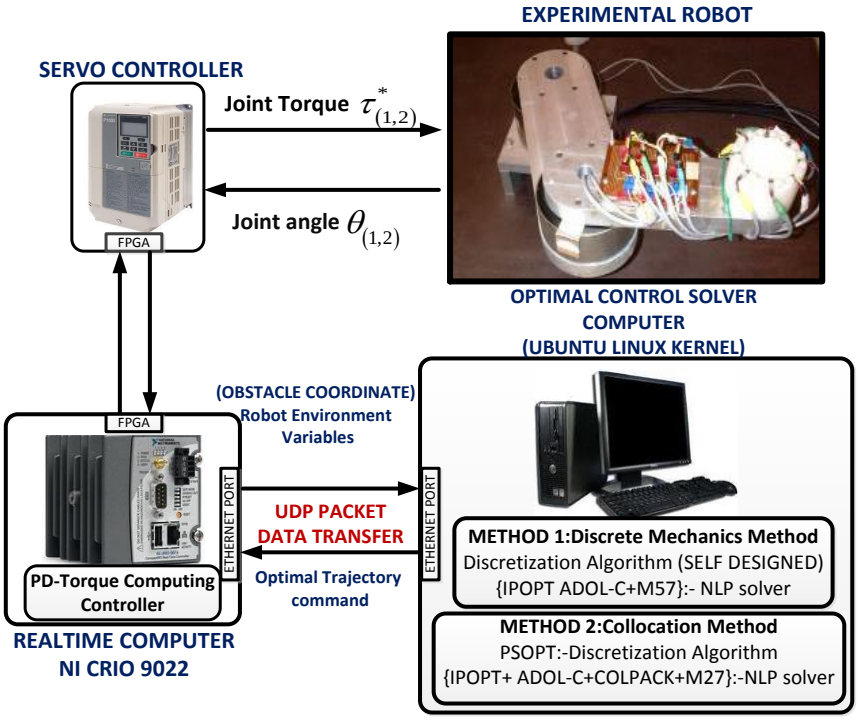

FIGURE 5.14: Control architecture for real-time trajectory generation.

5.3.3 Hierarchical Control Triggering

The individual activities carried out by the subsystems (Real-time controller and the Optimal Control solver) have to be properly ordered for successful motion coordination especially in scenarios where the robot interacts with dynamics environments. The

command for the robot has to preserve its collision-free attribute on arrival at the real-time controller from the optimal control solver as well and when implemented on the robot hardware.

The major information the solver needs for computing solutions are:

- The most recent state of the robot configuration in terms of joint poses i.e. (position, velocity).
- The coordinate position in the workspace of thee most recent obstacle locations through out the robot runtime.

Case structures ordering the activities of the networked systems ensures the motion spanning the robot is safe and proper triggering action is carried during runtime for various scenarios. The overall objective of the hierarchical control framework, is to ensure the 2-degree of freedom control structure is automatically switched between the stand-alone subsystems. Also it ensures safe motion coordination (collision free) is realized all through the solution phases from the initial to final desired position with respect to the problem setup using the appropriate case structures. The figure below illustrates the structure of hierarchical control employed to implement the optimal solutions as feed-forward for the experimental setup.

5.3.4 Torque Computing Controller (Workspace Position Control)

The general conceptual implementation is as follows: The solutions to the OCP are updated for use in the robot servo controller as the desired task-space trajectory. This corresponding control input sequence from the optimal trajectory solver is applied in feed forward. At a time instance "k", the robot uses knowledge of optimal trajectory of the most recent solution to a RHC problem and the environment variables to move the system along this optimal path as long as no parameter is violated such as collision with a dynamic obstacle as is the case of this study from its current state $x(k)$.

Based on the robots Lagrangian earlier formulated to represent the robots equation of motion and augmented with the suitable linear PD-control law, the corresponding torque actions for precisely moving the actuator of the links through the set of optimal

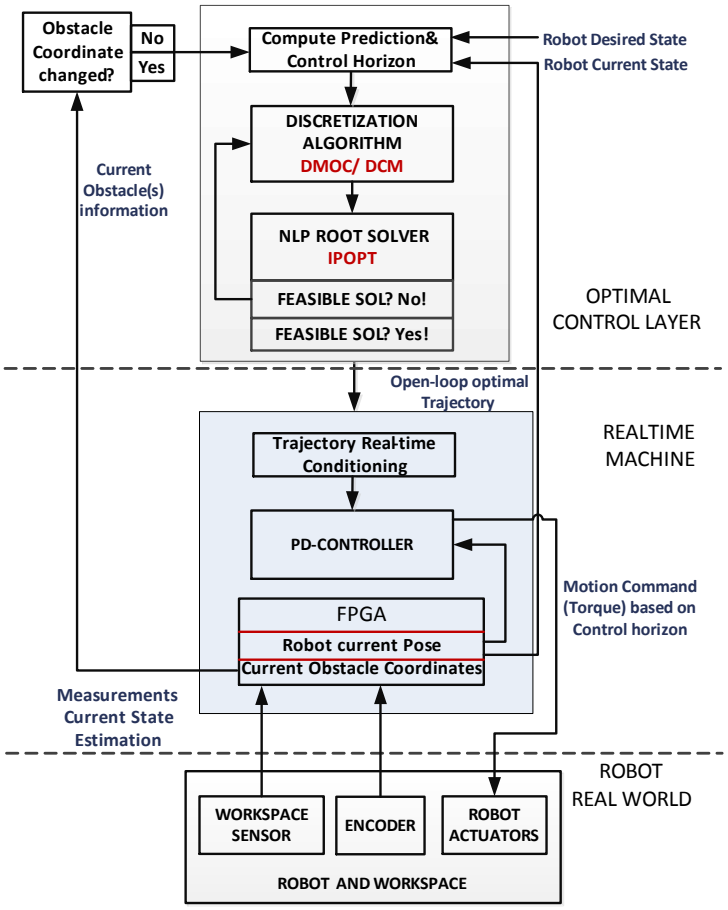

FIGURE 5.15: Hierarchical control system for the Experimental setup

states prescribing the desired optimal trajectory is formulated. This may be specified in the taskspace (joint coordinate) or workspace (Cartesian coordinate). For this work, the controller design is employed in the workspace. The major advantage of the workspace over the taskspace controller is the less computationally expensive formulation of the equations and the immunity the forward kinematics equation has over the inverse kinematics with respect to singularity points. The key subsystems framework of the two-degree of freedom control architecture showing the optimal control trajectory generation and low-level position controller is shown below.

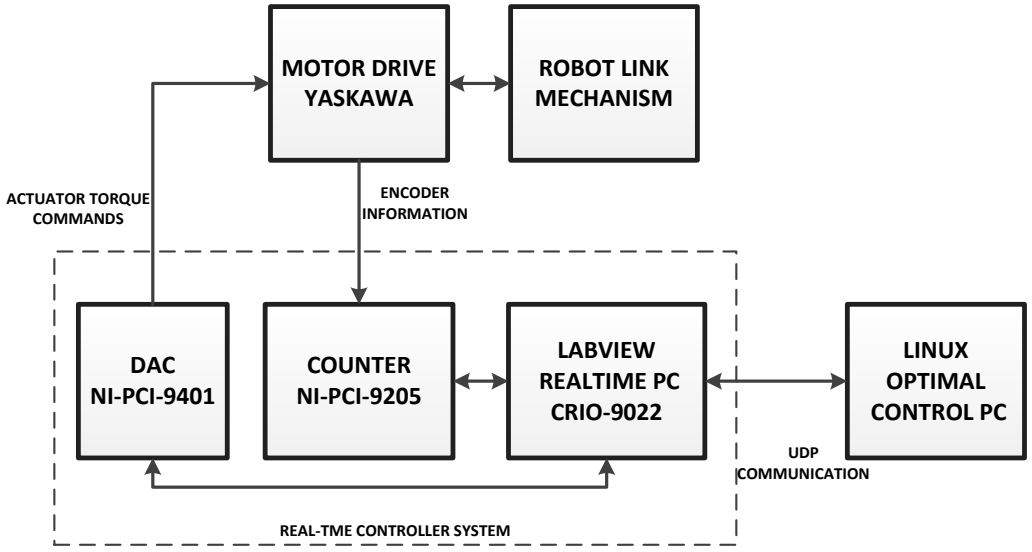

FIGURE 5.16: Key control subsystems interaction

With the dynamic parameters consistent to that earlier defined and as a vector of joint torque which is a function of time as a result of our time dependent end-effector trajectory, a control law in closed loop may be formulated accordingly. Following the formulation in [72], from the dynamic equation of the manipulator in 3.8, the P-D Torque computing controller is formulated as follows:

$$\tau_i(t) = M\left(q\left(t\right)\right)\ddot{q}\left(t\right) + C\left(q\left(t\right)\dot{q}\left(t\right)\right)\dot{q}\left(t\right) + g\left(q\left(t\right)\right) \tag{5.1}$$

Using the forward kinematic equations of the robot in question, optimal end-effector trajectory as x_d, y_d in the robot workspace coordinates are computed. Ignoring the gravity effect in the dynamics since our planar configuration KCM is immune to its effects; the dynamics equation by imploring the Jacobian function mapping the joint variables $q(t)$ to the Cartesian space $[x, y]$ may be rewritten as:

$$J\left(q\right) = \frac{\partial f}{\partial q} \text{ and } \dot{x} = J\left(q\right)\ddot{q}$$
$$\dot{q} = J^{-1}\dot{x} \text{ and } \ddot{q} = J^{-1}\ddot{x} + \frac{d}{dt}\left(J^{-1}\right)\dot{x}$$

Multiplying through and substituting these mapping equations, the dynamic equation may be rewritten as:

$$\tau_i(t) = J^T\left\{J^{-T}M(q(t))J^{-1}\ddot{x}(t) + \left[J^{-T}C(q(t)\dot{q}(t))J^{-1} + Mq(t)\frac{d}{dt}J^{-1}\right]\dot{x}(t)\right\} \tag{5.2}$$

A PD controller can then be incorporated at this stage to enable tracking of the desired Cartesian coordinate position by computing linear gains about the intermediate operating points for position control.

$$P_{gain} = K_p(q - q_d)$$
$$D_{gain} = K_d(\dot{q} - \dot{q}_d)$$

This can be defined as an error represented as the difference of the present position and the desired position $e = (q - q_d)$ multiplied by a scalar gain for the position and velocity. The time varying torque commands for position/trajectory control with the augmented PD control law in the workspace coordinates can be specified as:

$$\begin{aligned}\tau_i(t) = J^T\left[J^{-T}M(q(t))J^{-1}(\ddot{x}(t) - D_{gain} - P_{gain})\right] \\ +J^T\left[\left(J^{-T}C(q(t)\dot{q}(t))J^{-1} + Mq(t)\tfrac{d}{dt}J^{-1}\right)\dot{x}(t)\right]\end{aligned} \tag{5.3}$$

5.3.5 Experiment 1: Reference Set-point Problem

The reference set-point following problem of scenario-2 earlier formulated in the simulation section is implemented on the experimental 2-link robot setup employing the optimal desired trajectory as input commands for a Torque-computing controller. The appropriate torque commands for the actuators at the robot joints to steer the robot along this desired path are computed with the feedback PD-controller which is set to run in real-time at $2ms$. Using a time based interpolation dictated by the discretization process; the terminal time for the optimization problem is conditioned to run at the same rate as the real-time PD-controller in light of preserving the time properties of the optimal trajectory. The process is automated and the robot environmental variables (robot pose, obstacle coordinates and desired final pose) are communicated to the solver using the UDP connectionless data transfer. This data acts as a trigger for the solver

to compute the solutions and in the same way solutions (in form of desired optimal trajectories) are communicated back to the real-time machine as motion coordination commands.

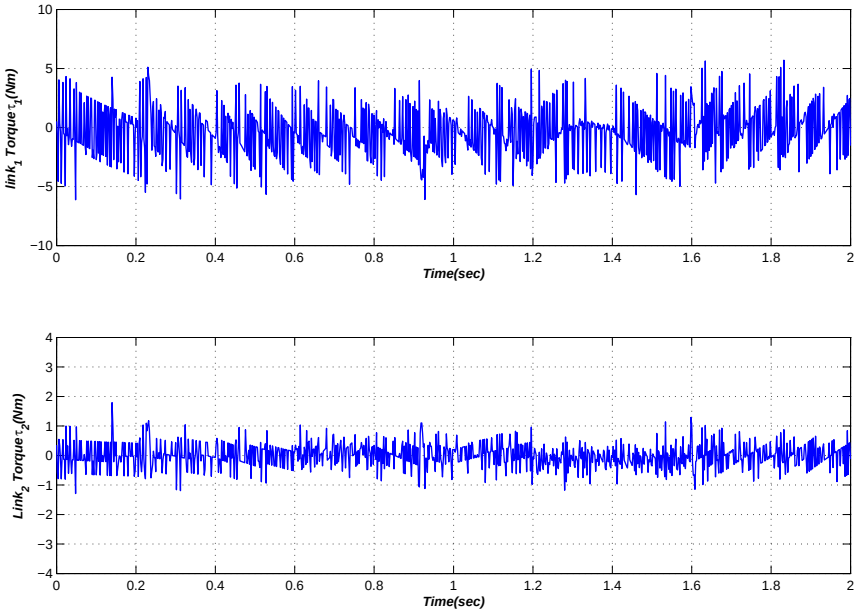

FIGURE 5.17: Closed-loop PD-controller performance for joints motors actuation.

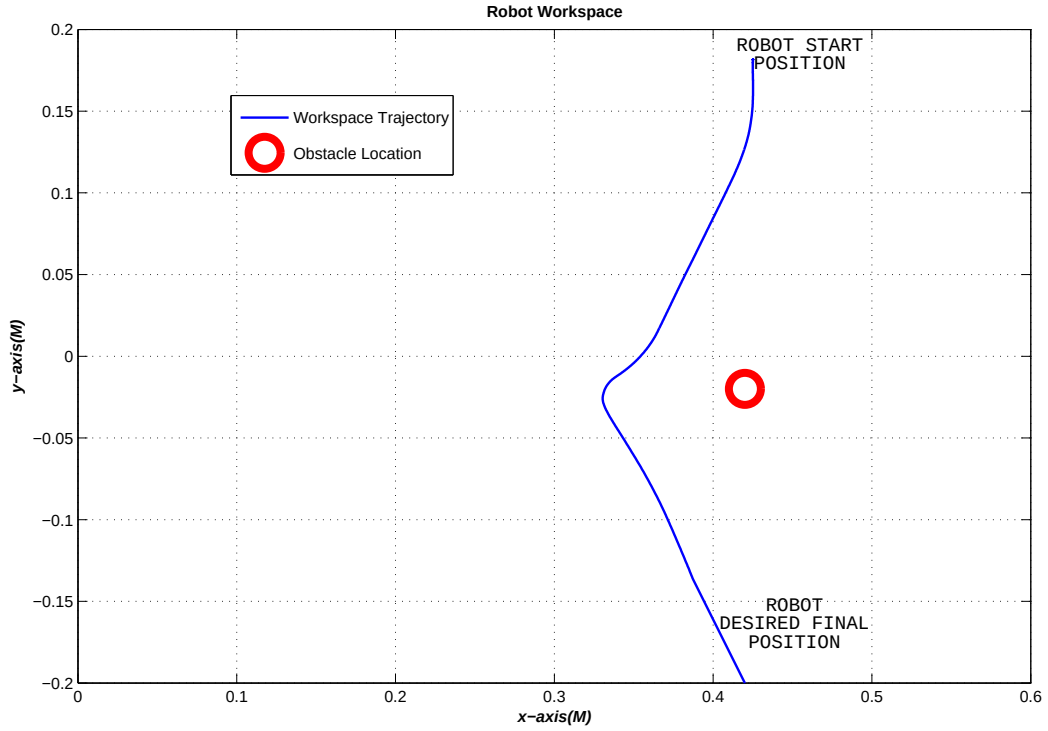

FIGURE 5.18: Robot optimal workspace trajectory.

5.3.6 Experiment 2: Dynamic Obstacle Avoidance Problem

A point-to-point motion with initial and final joint configurations of the robot links θ_1 and θ_2 as [0,pi/3] and [pi/4,pi/4] respectively is formulated. The optimal control problem is specified with a prediction horizon of 600 milliseconds and a control horizon of 300 milliseconds implemented using the RHC framework. The communication structure is consistent with that of experiment 1. Dynamically changing obstacle locations are simulated within the robot workspace at different time intervals along the robot initial trajectory and the manipulator is forced to maneuver around them reaching a desired final pose. The servo-controller real-time loop computing the torque commands for the link runs at $2ms$. The first obstacle with workspace coordinate location (0.43,0.15), is sensed at the inception of the robot motion with the obstacle changing during run time. Depending on the property of the obstacle, the control horizon may be corrected to accommodate for the implementation of the sub-optimal trajectories to ensure robot continuous collision-free motion.

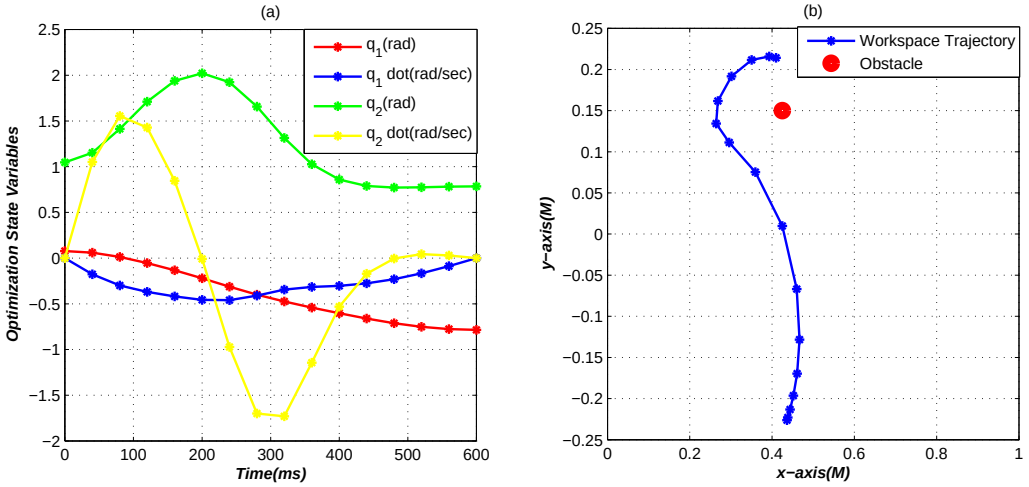

FIGURE 5.19: (a)Optimal states and (b)Optimal trajectory in the 1st RHC iteration.

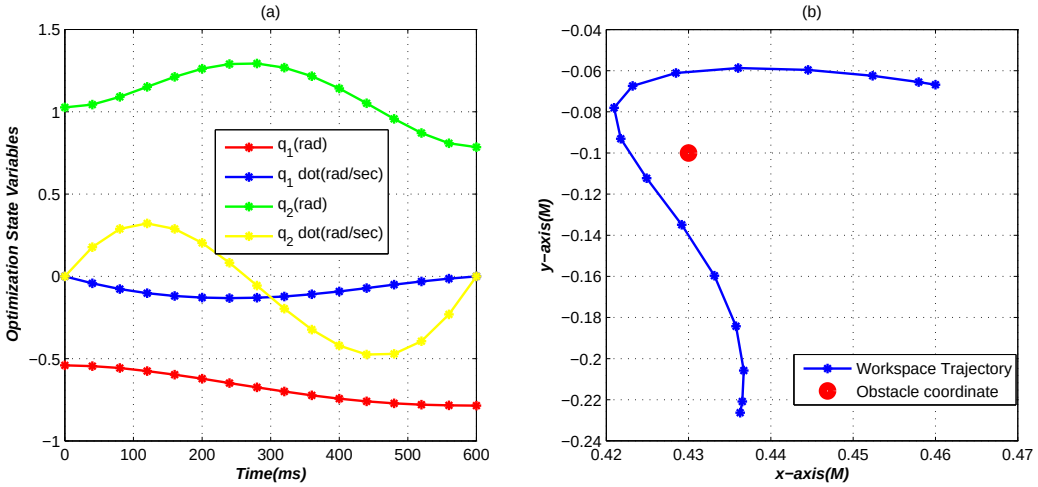

FIGURE 5.20: (a)Optimal states and (b)Optimal trajectory in the 2nd RHC iteration.

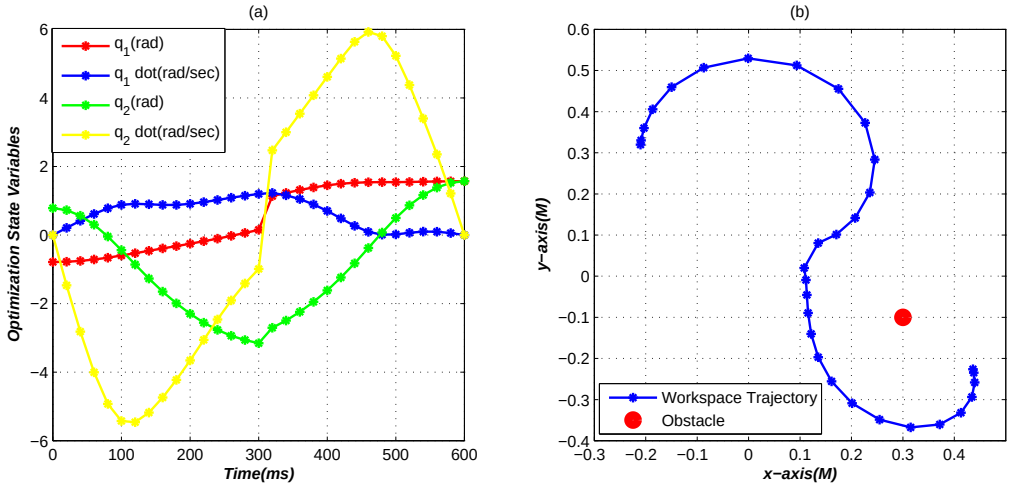

FIGURE 5.21: (a)Optimal states and (b)Optimal trajectory in the 3rd RHC iteration.

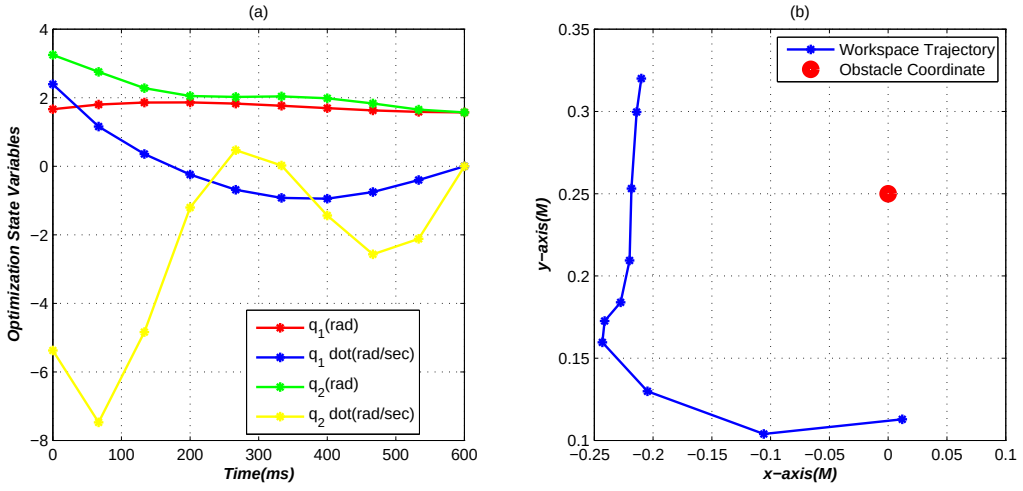

FIGURE 5.22: (a)Optimal states and (b)Optimal trajectory in the 4th RHC iteration.

Optimization Information	**Phase 1**	**Phase 2**	**Phase 3**	**Phase 4**
CPU time (seconds)	8.4e-02	1.9e-01	2.2e-01	4.3e-01
Optimal (unscaled) cost function value	1.21929e-02	1.38008e-01	1.97489e-01	1.0900e-01
Phase Initial time(seconds)	0	0	0	0
Phase final time(seconds)	0.6	0.6	0.6	0.6
Number of Iterations	50	46	22	14
Phase maximum relative local error	2.35123e-03	3.13981e-3	3.44141e-03	4.00651e-3
Number of objective function evaluations	32	32	32	32
Number of objective gradient/constraint Jacobian evaluations	31	31	31	31
Number of equality/inequality constraint evaluations	62	62	62	62
Number of Lagrangian Hessian evaluations	0	0	0	0
Interpolation points(Nodes)	16	16	16	16

TABLE 5.7: Optimization summary of solutions in RHC-phases of sub-optimal trajectories.

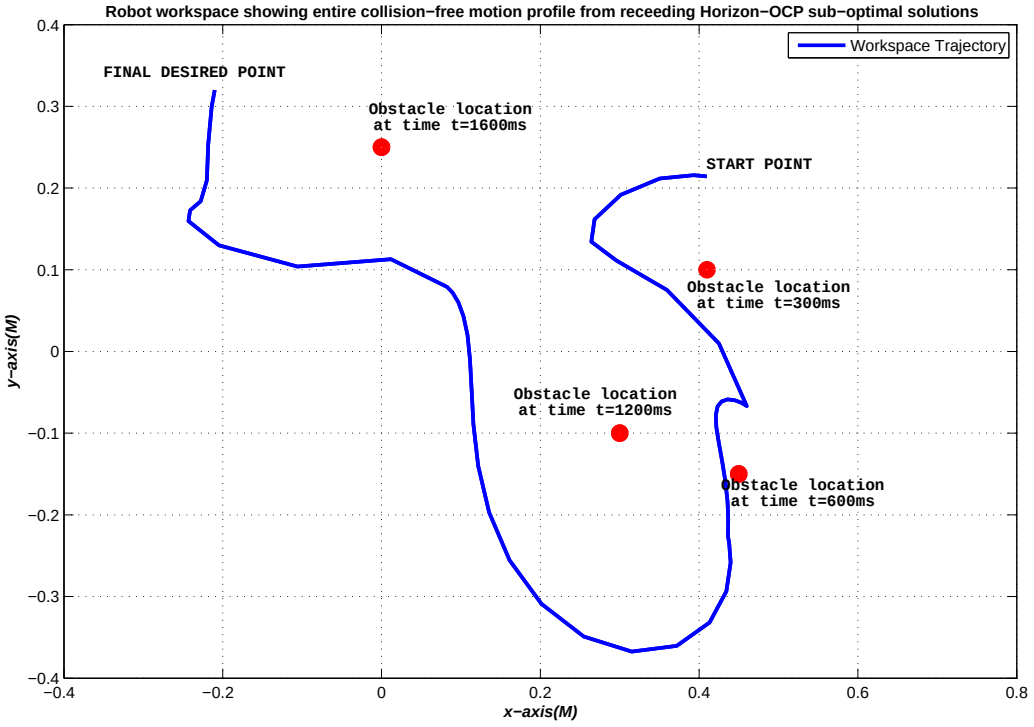

FIGURE 5.23: Generated optimal collision-free trajectory with time-varying obstacles.

Chapter 6

CONCLUSIONS

This chapter gives a general closure of the problem studied, the outcome from the concepts employed as stated in the preceding chapters, observations and conclusions from the results arrived at as well as insights for further work.

6.1 Observations, Limitations and Future Work

6.1.1 Conditioning the Solution

The initial guess is a crucial optimization information needed by the SQP algorithm to derive the root solutions of the continuous OCP in a NLP form. The accuracy of the solution depends largely on its choice. This is done by propagating the differential equation associated with the problem or is derived from intuition based on the prior information/knowledge of the probable solution of the problem in question.
Highly varying solutions were observed with very slight changes in the initial guess. This shows that the resulting NLP's after discretization of the OCP are very sensitive to this optimization parameter. Both the DMOC and DCM methods show high sensitivity to the initial guess and varied solution are observed with different values specified. A high variance from the best choice resulted in ill-conditioned solution as well.
Incorrect or infeasible solutions are also noticed when this is not within proper bounds (collocation points a too few or too many) for the problem. A realistic and intuitive remedy is manual fine tuning. This is however a limitation for the type of problem

investigated in this work (online/real-time trajectory generation). For an intended automated process such as this, the root finding algorithm computing solutions to OCP has to be given the appropriate initial guess at each iteration to compute the solutions. The problem setup in PSOPT cannot be redesigned to accommodate this crucial development because it gives no room for auto-generating the initial guesses beyond the first successive RHC problem phases and these constraints the quality of the solution when the scenario may not be recognized intuitively.

6.1.2 Quality of the Optimal Solutions (Accuracy)

The DCM discretization technique and its solutions at best, give approximate solutions for the optimal control and state trajectories. From the various scenarios, solutions are generally correct except for problems where the solution has a far varying nature within the phases, resulting to incorrect or infeasible solutions.
The DMOC suffers less in this regard but the wrong values of the configuration space discretization parameter may render the solution impractical.
For both methods, many variables and system parameters affect the quality. The users understanding of the problem is most important and choosing the most appropriate bounds and conditioning values will surely give reasonable results.

6.1.3 Handling Scenarios with Infeasible Solutions

Many conditions characterize the solution as feasible or during the optimization process. One important optimization based check for feasibility, is the error tolerance in the discretization size and interval. The discretization error ratio must fall between a certain range predefined by the user based on the mix of quadrature rules and approximation polynomial used prior to computation of the solution for the DCM. The NLP local solution tolerance in terms of the constraint bounds and scaling factors is also another condition in the SQP algorithm which cannot be violated else the solution is infeasible. This is also a predefined bound for the solver. This optimization conditions give information about the optimization process and foster the users understanding of the points where ill-conditioning occur; however, it is only desirable for problems with single phase solutions.

Also, scenarios with physically impossible instances resulted in infeasible solutions are expected. Such instances with respect to the problems investigated in this report are found when the initial condition imposed on the problem already violates the manipulator geometric constraints (obstacles within the space of the link enclosing ellipsoids) or cases in which the obstacle coordinates are located beyond the robot work envelope. The approach utilized to handle such instances was to define rules or case structures to check such occurrences and trigger the appropriate hierarchical action to resolve such illogical scenarios or ill-conditioned solutions. This ensured the collision-free properties of trajectories and also the integrity of the state and control parameters specified for the experimental test-bed robot motion commands as optimal.

6.1.4 Method Suitability for Online Optimization

Although the problem scale was amplified due to discretization of both the state and control variables as optimization parameters, the high sparsity in the Jacobian matrices of the NLP gave the SQP algorithm the advantage of achieving speedy computation in arriving at the local minimal solutions application for real-time motion profiling.
This discretization technique in the Collocation methods at best, give approximate solutions for the optimal control and state trajectories. From the various scenarios, solutions are generally correct except for problems where the solution has a far varying nature within the phases, resulting to incorrect or infeasible solutions.
The data packets sent by the UDP-IP also aided the interaction between the Solver-PC and the Real-time machine and proper activity ordering ensured the bi-directional communication was instantaneous.

6.1.5 Choice of Discretization Interval Points (Nodes)

The solution accuracy and computation speed depends on the choice of the discretization points (nodes) with respect to both OCP methods implemented as shown in the simulation results of chapter 5. For instance, where full knowledge of the problem is not known prior to solving it, and assigning the best choice of discretization points cannot be derived intuitively, generated solutions may give non-feasible or ill conditioned solutions. Manually generating the nodes is undesirable and is a non-trivial limitation, as

the hierarchical control system runs in an automated field.
The results show that the DMOC algorithm results in a more immune NLP with respect to choice of discretization points used unlike the collocation method. Ill conditioned solutions are observed when the number of nodes is far from a convergence range suitable for the problem setup as well as the collocation method further exhibiting a much larger variance in resulting solutions for just a slight variation in the number of collocation points.

6.1.6 State and Control Boundaries

Both methods (DCM and DMOC) handled state and control constraints perfectly. The values for this optimization parameter were incorporated based on the hardware setup for all the problem formulations as depicted in the simulation results shown in chapter 5.
Both methods only impose the constraint sets at the nodes, and not within the interpolation points. This maybe an issue for cases where constraints need to be active all through the phase of the problem implementation and so, could be a hindrance for smooth collision-free trajectory generation problems. The approach to circumvent this problem will be to use as many as is suitable discretization points at the expense of computation size and solution time.

6.1.7 Inactive Path constraint: Passive Links Strategy

To further speed-up the computation process and still preserve the accuracy of the solutions, the geometric state of the KCM links with respect to the orientation of obstacle coordinates dictates the path constraint on the individual manipulator links as inactive or not. This was done by checking the state of the links minimum enclosing ellipsoids with respect to the obstacle coordinate and in essence ignoring the algebraic constraints generated by obstacles when they did not physically constrain the motion of the link in question. In a case where an obstacle did not interfere with all the links, there was a faster computation of optimal solutions due to considerably less constraint sets on the problem.

6.1.8 Incorporating Workspace Vision Sensors

For the experimental setup in this work, the obstacle locations were simulated at arbitrary time instances into the OCP directly from the real-time computer. A position sensor was not incorporated into the robot arm hardware. More practical investigations may suffice by mounting an active sensory device (e.g Camera) in the robot workspace for monitoring these arbitrary changing obstacles and enhance more robust real-time operations.

6.2 Conclusions

An high level summary of this work maybe described as follows:

- Design and implementation of two direct optimal control approaches for generating collision-free trajectories in KCM robot systems where arbitrarily moving obstacles may exist in the robot work envelope for point-to-point motion and where predefined time-varying paths were prior specified.
- Implementing the real-time OCP using a conceptual control framework: The Receding Horizon Control scheme as platform to realize online and real-time motion control capturing the dynamic properties of the time-varying obstacles.
- Design of a two-degree-of-freedom control platform. A Hierarchical Control System equipped through bi-directional UDP communication was implemented to interface two independent computer processes for enhanced real-time interaction and triggering control for activity ordering of the overall motion control system. The real-time performance of the servo-controller was assured and the speed of the optimal control algorithm generating the collision-free trajectories enhanced as well.
- Successful validation of the process above on an experimental two-link KCM using a linear feedback PD-controller to compute torque commands for steering the robot along the specified input set-points of collision-free optimal trajectories.

This work shows the success and prospects of using direct optimal control methods (DMOC and DCM) for trajectory generation problems in real-time applications for high

level motion planning in Kinematic Chain Manipulator robots which may be extended to related constrained mechanical systems.

Bibliography

[1] Hoam chung and Soo Jeon. Collision-free trajectory generation of robotic manipulators using receding horizon strategy. *American Control Conference (ACC), IEEE*, pages 1692–1697, 2011.

[2] World Robotics 2012 Industrial Robots. Industrial robot statistics. *International federation of robotics*, 2, 2012. URL `www.ifr.org/uploads/media/WR_Industrial_Robots_2012_Executive_Summary.pdf`.

[3] Steven M.LaValle. Planning algorithms. *Cambridge University Press*, 2006.

[4] J. E. Bobrow, S. Dubowaky, and J. S. Gibson. On the optimal control of robotic manipulators with actuator constraints. *American Control Conference*, pages 782–787, 1983.

[5] Florin Moldoveanu, Vasile Comnac, Dan Floroian, and Cristian Boldisor. Trajectory tracking control of a two-link robot manipulator using variable structure system theory. *Journal of Control Engineering and Applied Informatics*, 7(3):56–62, 2005.

[6] M.E. Kahn, A.K. Bejczy, and B. Roth. The near-minimum time control of open-loop articulated kinematic chains. *Stanford Artificial Intelligence Memo*, 1:164–172, 1971.

[7] R. Paul. Modeling and trajectory calculations and servoing of a computer controlled arm. *Stanford Artificial Intelligence Memo*, 117(1), 1972.

[8] J-C. Latombe L.E Kavraki, P.Svestka and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configurations spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[9] J.H Yakey S.M Lavalle and L.E Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 12:1671–1676, 1999.

[10] E.Kim K.Oh, J.P.Hwang and H.Lee. Path plannig of a robot manipulator using retieval rrt strategy. *World Academy of Science, Egineering and Technology*, 12, 2007.

[11] Balint Kiss Emese Szadeczky-Kardoss. Extension of the rapidly-exploring random trees algorithm with key configurations for nonholonomic motion planning. pages 293–309, 2007.

[12] J.J Kuffer S.M Lavalle. Rapidly-exploring random trees:a new tool for path planning. *Technical report. Computer Science Dept,Iowas State University*, (11):88, Oct 1998.

[13] J.J Kuffer S.M Lavalle. Rapidly-exploring random trees:progress and prospect. *Internationsl workshop on Algorithmic and computational Robotics(WAFR)*, pages 293–309, 1999.

[14] Masato Suzuki Sejiji Aoyagi Chikatoyo Nagata, Eri Sakamoto. Path generation and collision avoidance of robot manipulators for unknown moving obstacles using real-time rapidly-exploring random trees (rrt). *Service Robotics and Mechatronics*, 2007.

[15] Oussama Khatib Oliver Brock. Real-time obstacle avoidance and motion coordination in a multi-robot workcell. *IEEE Symposium on Assembly and Task Planning*, July 1999.

[16] Oussama Khatib Oliver Brock. Real-time replanning in high-dimensional configuration spaces using sets of homotopic paths. *IEEE International Conference on Robotics and Automation*, April 2000.

[17] Nicholas Petit, Mark B. Milam, and Richard M. Murray. Inversion based constrained trajectory optimization. *5th IFAC symposium on nonlinear control systems*, 2001.

[18] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Proceedings of the 16th annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[19] Narendra Karmarkar. New parallel architecture for sparse matrix computation based on finite projective geometry. *Proceedings of the 1991 ACM/IEEE Conference*, pages 358–369, 1991.

[20] Andreas Wachter and Lorenz T Biegler. On the implementation of an interior point filter line search algorithm for large scale nonlinear programming. *Mathematical Programming*, pages 25–57, 2006.

[21] Anthony V. Fiacco and Garth P. McCormick. Nonlinear programming: Sequential unconstrained minimization techniques. *Unabridged, corrected republication*, 1990.

[22] Wassem Ahmed Kamal, Da wei gu, and Ian Postlethwaite. Realtime trajectory planning for uavs using milp. *44th IEEE Conference on Control and Automation*, pages 3381–3386, 2005.

[23] Cedric S. Ma and Robert H. Miller. Optimal path planning for real-time applications northrop grumman integrated systems. *American Control Conference, IEEE Control Conference*, page 6, 2006.

[24] Arthur Ricards and Jonathan P. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. *American Control Conference*, pages 1936–1941, 2002.

[25] Hao Ding, Gunther Reiig, and Olaf Stursberg. Increasing efficiency of optimization-based path planning for robotic manipulators. *Decision and Control and European Control Conference CDC-ECC*, pages 1399–1404, 2011.

[26] Hao Ding, Mingxiang Zhou, and Olaf Stursberg. Optimal motion planning for robotic manipulators with dynamic obstacles using mixed-integer linear programming. *Control and Automation*, pages 934–939, 2009.

[27] Paulo Tabuada and George J . Pappas. Hierarchical trajectory generation for a class of nonlinear systems. *Automatica*, pages 701–708, 2005.

[28] J. E. Bobrow, F. C. Park, and A. Sideris. Recent advances on the algorithmic optimization of robot motion. *Fast Motions in Biomechanics and Robotics*, pages 21–41, 2006.

[29] Michael J. Van Nieuwstadt and Richard Murray. Real-time trajectory generation for differentia flat systems. *California institute of Technology*, 1997.

[30] Elijah Polak, Hoam Chung, and S. Shankar Sastry. An external active-set strategy for solving optimal control problems. *Automatic Control, IEEE Transactions*, 54 (5):1129–1133, 2009.

[31] A. H. Levis, S. I. Marcuc, W. R. Perkins, P. Kokotovic, M. Athans, R. W. Brockett, and A. S. Willsky. Challenges to control: A collective view. *Automatic Control, IEEE Transactions*, 32(4):275–285, September 1987.

[32] R. Johansson. Quadratic optimization of motion coordination and control. *Automatic Control, IEEE Transactions on Robotics and control*, pages 1197–1208, 1990.

[33] A. S. Morris and A. Madani. Quadratic optimal control of a two-flexible-link robot manipulator. *Robotica*, 16(1):97–108, 1996.

[34] Tamer Basar Francesco Bullo Gregory J. Toussaint. Motion planning for nonlinear underactuated vehicles using h-infinity techniques. *IEEE American Control Conference*, pages 4097–4103, 2001.

[35] Sigurd Skogestad and Ian Postlethwaite. Multivariable feedback control. *Analysis and Design (2nd Edition)*, 2005.

[36] Dan Simon. Optimal state estimation: Kalman, h-infinity, and nonlinear approaches. *Wiley-Interscience*, 2006.

[37] Basar Tamer and Bernhard Pierre. H-infinity optimal control and related minimax design problems. 2008.

[38] R. F. Hartl, S.P. Sethi, and R.G. Vickson. A survey of the maximum principles for optimal control problems with state constraints. *SIAM Review*, 37(2):181–218, 1995.

[39] R. Bulirsch, E. Nerz, H. J. Pesch, and O. Von Stryk. Combining direct and indirect methods in optimal control: Range maximization of a hang glider. *Mathematics Institute*, 1991.

[40] H. J. Pesch. A practical guide to the solution of real-life optimal control problems. *Control and Cybernetics*, 23(1/2):7–60, 1994.

[41] L.S. Pontryagin, V.G. Boltyanskii, R.V. Gamkrelidze, and E.F. Mishchenko. The mathematical theory of optimal processes. *Control and Cybernetics John Wiley & Sons, New York*, 1962.

[42] B. S. Chen, T.-S. Lee, and J.-H. Feng. A nonlinear h-infinity control design in robotics systems under parametric perturbation and external disturbance. *International Journal of Control*, pages 439–461, 1994.

[43] J. Park and W. K. Chung. Analytic nonlinear h-infinity inverse-optimal control for euler-lagrange system. *IEEE Transactions on Robotics and Automation*, 16(6): 847–854, 2000.

[44] H. Gorecki, A. Korytowski, M. Symkat, and A. Turnau. Optimal control- a survey. 2004.

[45] C.Hargraves and S.Paris. Direct trajectory optimization using nonlinear programming and collocation. *AIAA Journal of Guidance, Control and Dynamics*, 10(4), 2012.

[46] Hermann Brunner. Collocation methods for volterra integral and related functional equations. *Cambridge Monographs on Applied and Computational Mathematics*, 2004.

[47] W. Murray, P. E. Gill, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12(4):979–1006, 2002.

[48] E. Polak. Optimization: Algorithms and consistent approximations. *Applied Mathematical Sciences*, 7, 1997.

[49] On the convergence of a sequential quadratic programming method with an augmented lagrangian line search function. *Optimization*, 14(2):197–216, 1983.

[50] Michael. A. Patterson and Anil V. Rao. Exploiting sparsity in direct collocation pseudospectral methods for solving optimal control problems. *University of Florida. Gainesville, FL 32611*, 49(2), 2012.

[51] Jesse A. Pietz. Pseudospectral collocation methods for the direct transcription of optimal control problems. *Masc Thesis, Rice University*, 2004.

[52] Sina Ober-Blobaum, Oliver Junge, and Jerrold E. Marsden. Discrete mechanics and optimal control: An analysis. *California Institute of Technology*, 2008.

[53] J. E. Marsden and M. West. Discrete mechanics and variational integrators. *Actra Numerica*, 10(357):514, 2001.

[54] Ari Stern and Mathieu Desbrun. Discrete geometric mechanics for variational time integrators. *Caltech*, 2010.

[55] Jaiwey Zhang, Weizhong Zhang, and Jiayuan Shan. A new dmoc-based approach to solve goddard rocket problem. *IEEE conference*, pages 2165–2169, 2012.

[56] Weizhong Zhang and Tamer Inanc. A tutorial for applying dmoc to solve optimization control problems. *IEEE Conference on Robotics and Control Systems*, pages 1857–1862, 2010.

[57] Weizhong Zhang, Tamer Inanc, and Jerrold E. Marsden. Dmoc approach to real-time trajectory generation for mechanical systems. *IEEE Conference on Control, Automation, Robotics and Vision*, pages 2192–2195, 2008.

[58] Francois Margot Yoshiaki Kawajir and Andreas Wachter. Introduction to ipopt: A tutorial for downloading, installing, and using ipopt. 2010.

[59] W. Murray P. E. Gill and M. A. Saunders. Snopt: An sqp algorithm for largescale constrained optimization. *Report NA 97-2, Department of Mathematics, University of California, San Diego USA*, 1997.

[60] S. P. Han. Superlinearly convergent variable-metric algorithms for general nonlinear programming problems. *Mathematical Programming*, page 11:263282, 1976.

[61] J.M.Maiejowski. Predictive control with constraints. *Prentice Hall*, 1 edition, September 2000.

[62] Christopher V. Roa, Stephen J. Wright, and James B. Rawlings. On the application of interior point methods to model predictive control. *Journal of optimization theory and application*, 99(3):723–757, 1998.

[63] L.C. Rabelo. Intelligent control of a robotic arm using hierarchical neural network systems. *IEEE Seattle International Joint Conference on Automation*, 2:747–751, 1991.

[64] Elon Rimon and Stephen P. Boyd. Obstacle collision detection using best ellipsoid fit. *Journal of intelligent and robotic systems*, 18(2):105–126, 1997.

[65] N. Moshtagh. Minimum volume enclosing. *Convex Optimization*, 2005. URL `www.mathworks.com/matlabcentral/fileexchange/9542`.

[66] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). *New results and new trends in computer science*, pages 359–370, 1991.

[67] B. P. Burrell and M. J. Todd. The ellipsoid method generates dual variables. *Mathematics of Operations Research*, 10(4):688–700, 1985.

[68] P. Sun and R. M. Freund. Computation of minimum volume covering ellipsoids. *Operations Research*, pages 690–706, 2004.

[69] Victor M. Becerra. Psopt optimal control solver user manual. *University of Reading, Release version 3*, 2011.

[70] Von stryk O. and R. Bulirsch. Direct and indirect methods for trajectory optimization. *Annals of Operations Research*, 37(1):357–373, 1992.

[71] Mark .W Spong, Seth Hutchinson, and Mathukumalli Vidyasagar. Robot modeling and control. *Hoboken New Jersey: John Wiley & Sons*, 2006.

[72] S. S. Sastry S. Shankar Sastry Richard M. Murray, Zexiang Li. A mathematical introduction to robotic manipulation. *CRC Press*, 1994.

Appendix

This appendix includes the full code that may be used to reproduce the results for the 2-link and 3-link Robot manipulator solved in chapter 4 and implemented in the experimental setup in chapter 5 for the DCM and DMOC methods.

.1 DCM Code

PSOPT: A Tool for Implementing Optimal Control problems by Direct Collocation methods.

PSOPT [69] is a C++ based library which implements the Direct Collocation methods explained in chapter 3. The resulting nonlinear programming problem is solved using an Interior Point Optimization (IPOPT) software library IPOPT; which is a sequential programming algorithm. IPOPT uses dependency solvers to compute the gradients, hessians and jacobian matrices and primarily the option used here is the Automatic Differentiator for C++; (ADOL-C). Synchronized use of the libraries: PSOPT/IPOPT is setup to solve optimal control problems in the following way consistent with the setup in the PSOPT user manual [69].
C++ code of the Optimal control Implementation for motion planning problem using PSOPT/IPOPT is as follows:

```
//c++ implemetation for 2-link Problem setup in PSOPT
        #include "psopt.h"
        #include "PracticalSocket.h"
        #include <iostream>
        #include <cstdlib>
        #include <stdlib.h>
        #include <string.h>
        #include <pthread.h>
        #include <time.h>

const double xd=0.43, yd=0.0;//obstacle coordinate position
const double l1=0.32,l2=0.21,r1=0.16,r2=0.083545;//links and ellipse center
double q1i=0.0, q2i=pi/3, dq1i = 0.0, dq2i = 0.0; //initial-position&velocity
```

```
double dq1f =0.0, dq2f=0.0;//final position&velocity
const double alpha=0.835783364,beta=0.05194688,delta=0.028067364;//lumped
const double as1=0.065957,bs1=0.0088631,as2=0.030155,bs2=0.0041179; //ellipse
const double i_time=0, f_time=2,col_nodes =12,obstacle_c =2, i_path=0.0,
    f_path=200.0;
const double wu=1, we=100;

//  Define the end point (Mayer) cost function
adouble endpoint_cost(adouble* initial_states, adouble* final_states,adouble*
    parameters,adouble& t0, adouble& tf,adouble* xad, int iphase)
    {
    return 0.0;
    }
//  Define the integrand (Lagrange) cost function
adouble integrand_cost(adouble* states, adouble* controls, adouble*
    parameters, adouble& time, adouble* xad,int iphase)
    {
        adouble retval;
    adouble u1 = controls[CINDEX(1)];//controls
        adouble u2 = controls[CINDEX(2)];
        adouble q1    = states [CINDEX(1)];//states
        adouble q1dot = states [CINDEX(2)];
        adouble q2    = states [CINDEX(3)];
        adouble q2dot = states [CINDEX(4)];
        adouble xr =l1+l2/2, yr = -l2*1.732213/2*cos(pi*(time/2 - 1));
        adouble xtip = l1*cos(q1) + l2*cos(q1+q2);
        adouble ytip = l1*sin(q1) + l2*sin(q1+q2);
        adouble x=xtip,y=ytip;
        retval =0.5*(
    (we*(x-xr)*(x-xr))+we*(y-yr)*(y-yr))+0.5*wu*((u1*u1)+(u2*u2));
        return retval;
        }
//  Define the dynamic equations of the robot
void dae(adouble* derivatives, adouble* path, adouble* states, adouble*
    controls, adouble* parameters, adouble& time,adouble* xad, int iphase)
        {
        adouble q1    = states [CINDEX (1)];
        adouble q1dot = states [CINDEX (2)];
```

```
    adouble q2    = states [CINDEX (3)];
    adouble q2dot = states [CINDEX (4)];
    adouble u1 = controls[ CINDEX(1) ];
    adouble u2 = controls[ CINDEX(2) ];
    adouble M[2][2],C[2][2],Mi[2][2];
    M[0][0] = alpha+2.0*beta*cos(q2);
    M[0][1] = delta+beta*cos(q2);
    M[1][0] = M[0][1];
    M[1][1] = delta;
    adouble D = M[0][0]*M[1][1] - M[0][1]*M[1][0];
    D=1.0/D;
    Mi[0][0] =  D*M[1][1];
    Mi[0][1] = -D*M[0][1];
    Mi[1][0] = -D*M[1][0];
    Mi[1][1] =  D*M[0][0];
    C[0][0] = -q2dot*beta*sin(q2);
    C[0][1] = -(q1dot+q2dot)*beta*sin(q2);
    C[1][0] = q1dot*beta*sin(q2);
    C[1][1] = 0.0;
    adouble qdd[2];
    qdd[0] = Mi[0][0]*( u1-C[0][0]*q1dot-C[0][1]*q2dot ) + Mi[0][1]*(
u2-C[1][0]*q1dot-C[1][1]*q2dot );
    qdd[1] = Mi[1][0]*( u1-C[0][0]*q1dot-C[0][1]*q2dot ) + Mi[1][1]*(
u2-C[1][0]*q1dot-C[1][1]*q2dot );
    adouble q1dd = qdd[0], q2dd = qdd[1];
    derivatives [ CINDEX(1) ] = q1dot;
    derivatives [ CINDEX(2) ] = q1dd;
    derivatives [ CINDEX(3) ] = q2dot;
    derivatives [ CINDEX(4) ] = q2dd;
    adouble c1[1],c2[1];
    c1[0] = r1*cos(q1);
    c1[1] = r1*sin(q1);
    c2[0] = l1*cos(q1)+r2*cos(q1+q2);
    c2[1] = l1*sin(q1)+r2*sin(q1+q2);
    adouble p1 = q1;
    adouble p2 = q1+q2;
    path[0]=((pow((xd-c1[0])*cos(q1)-(yd-c1[1])*sin(q1),2)/as1)+...
    (pow((yd-c1[1])*cos(q1)-(xd-c1[0])*sin(q1),2)/bs1))-1;
```

```
        path[1]=((pow((xd-c2[0])*cos(p2)-(yd-c2[1])*sin(p2),2)/as2)+...
        (pow((yd-c2[1])*cos(p2)-(xd-c2[0])*sin(p2),2)/bs2))-1;
        }

//  Define the events function

void events(adouble* e, adouble* initial_states, adouble*
    final_states,adouble* parameters,adouble& t0, adouble& tf, adouble*
    xad,int iphase)
        {
        adouble q1i  = initial_states[ CINDEX (1)];
        adouble dq1i = initial_states[ CINDEX (2)];
        adouble q2i  = initial_states[ CINDEX (3)];
        adouble dq2i = initial_states[ CINDEX (4)];

        adouble q1f  = final_states[ CINDEX (1)];
        adouble dq1f = final_states[ CINDEX (2)];
        adouble q2f  = final_states[ CINDEX (3)];
        adouble dq2f = final_states[ CINDEX (4)];

    e[ CINDEX(1)]  = q1i;
        e[ CINDEX(2)]  = dq1i;
        e[ CINDEX(3)]  = q2i;
        e[ CINDEX(4)]  = dq2i;
        e[ CINDEX(5)]  = dq1f;
        e[ CINDEX(6)]  = dq2f;
        }
//  Define the phase linkages function
        void linkages( adouble* linkages, adouble* xad )
        {

        }
// main subroutine

int main(void)
        {
//  Declare key structures
    Alg  algorithm;
```

```
        Sol  solution;
    Prob problem;
//  Register problem name
        problem.name = "Two link planar robot arm";
    problem.outfilename = "twolink.txt";

//  Define problem level constants & do level 1 setup

    problem.nphases   = 1;
    problem.nlinkages = 0;
    psopt_level1_setup(problem);

//  Define phase related information & do level 2 setup

    problem.phases(1).nstates   = 4;
    problem.phases(1).ncontrols = 2;
    problem.phases(1).nevents   = 6;
    problem.phases(1).npath     = obstacle_c;
    problem.phases(1).nodes     = col_nodes;
    psopt_level2_setup(problem, algorithm);

//  Enter problem bounds information(boundaries on states and controls)

    problem.phases(1).bounds.lower.states(1) = -100.0;
    problem.phases(1).bounds.lower.states(2) = -100.0;
    problem.phases(1).bounds.lower.states(3) = -100.0;
    problem.phases(1).bounds.lower.states(4) = -100.0;
    problem.phases(1).bounds.upper.states(1) = 100.0;
    problem.phases(1).bounds.upper.states(2) = 100.0;
    problem.phases(1).bounds.upper.states(3) = 100.0;
    problem.phases(1).bounds.upper.states(4) = 100.0;
    problem.phases(1).bounds.lower.controls(1) = -30.0;
    problem.phases(1).bounds.lower.controls(2) = -6.0;
    problem.phases(1).bounds.upper.controls(1) = 30.0;
    problem.phases(1).bounds.upper.controls(2) = 6.0;
        problem.phases(1).bounds.lower.events(1)  = q1i;
    problem.phases(1).bounds.lower.events(2)  = dq1i;
//  Enter problem bounds information(events, terminal time and obstacles path)
```

```
    problem.phases(1).bounds.lower.events(3)  = q2i;
    problem.phases(1).bounds.lower.events(4)  = dq2i;
    problem.phases(1).bounds.lower.events(5)  = dq1f;
    problem.phases(1).bounds.lower.events(6)  = dq2f;
        problem.phases(1).bounds.upper.events =
    problem.phases(1).bounds.lower.events;
        problem.phases(1).bounds.lower.StartTime = i_time;
    problem.phases(1).bounds.upper.StartTime = i_time;
    problem.phases(1).bounds.lower.EndTime   = f_time;
    problem.phases(1).bounds.upper.EndTime   = f_time;
    problem.phases(1).bounds.lower.path(1) = i_path;
    problem.phases(1).bounds.lower.path(2) = i_path;
    problem.phases(1).bounds.upper.path(1) = f_path;
    problem.phases(1).bounds.upper.path(2) = f_path;

// Register problem functions //

    problem.integrand_cost      = &integrand_cost;
    problem.endpoint_cost       = &endpoint_cost;
    problem.dae                 = &dae;
    problem.events              = &events;
    problem.linkages    = &linkages;

//  Define & register initial guess

        int nnodes = problem.phases(1).nodes(1);
    DMatrix state_guess    =  zeros(4,nnodes);
    DMatrix control_guess  =  zeros(2,nnodes);
    DMatrix time_guess     =  linspace(0,f_time,nnodes);
    state_guess(1,colon()) = linspace(q1i,-pi/6,nnodes);
    state_guess(2,colon()) = linspace(dq1i,0.0,nnodes);
    state_guess(3,colon()) = linspace(q2i,pi/3,nnodes);
    state_guess(4,colon()) = linspace(dq2i,0.0,nnodes);

    problem.phases(1).guess.states   = state_guess;
    problem.phases(1).guess.controls = control_guess;
    problem.phases(1).guess.time     = time_guess;
```

```
//  Enter algorithm options (choose the quadrature rules/ and polynomials for
    ODE approximation. change to psuedospectral ones depending on problem)

    algorithm.nlp_iter_max                = 1000;
    algorithm.nlp_tolerance               = 1.e-3;
    algorithm.nlp_method                  = "IPOPT";
    algorithm.collocation_method              = "trapezoidal";
    algorithm.ode_tolerance               = 1.e-3;

    algorithm.derivatives = "automatic";
    algorithm.diff_matrix  = "central-differences";
    algorithm.defect_scaling = "jacobian-based";
    algorithm.mr_max_increment_factor = 0.01;
    algorithm.mr_max_iterations = 1;
    algorithm.hessian = "exact";
        algorithm.scaling = "automatic";
    algorithm.mesh_refinement  = "automatic";

//  Call PSOPT to solve the problem and extract solutions

    psopt(solution, problem, algorithm);
    DMatrix x = solution.get_states_in_phase(1);
    DMatrix u = solution.get_controls_in_phase(1);
    DMatrix t = solution.get_time_in_phase(1);
    DMatrix l = solution.get_dual_costates_in_phase(1);
//  plotting
    DMatrix x1 = l1*cos(x(1,colon()));//link 1 tip
    DMatrix y1 = l1*sin(x(1,colon()));
        DMatrix x2 =
    l1*cos(x(1,colon()))+l2*cos(x(1,colon())+x(3,colon()));//end-effector
    DMatrix y2 = l1*sin(x(1,colon()))+l2*sin(x(1,colon())+x(3,colon()));
        DMatrix xe1 = l1/2.0*cos(x(1,colon())); //ellipse 1 center
    DMatrix ye1 = l1/2.0*sin(x(1,colon()));
        DMatrix xe2 =
    l1*cos(x(1,colon()))+r2*cos(x(1,colon())+x(3,colon()));//ellp center
    DMatrix ye2 = l1*sin(x(1,colon()))+r2*sin(x(1,colon())+x(3,colon()));
        DMatrix ang1 = x(1,colon());
```

```
        DMatrix ang2 = x(3,colon());
    DMatrix alpha = colon(0.0, pi/20, 2*pi);
        DMatrix xObs1 = sqrt(1.0e-5)*cos(alpha) +xd;//obstacle coordinate
    "circle"
    DMatrix yObs1 = sqrt(1.0e-5)*sin(alpha) +yd;
        x.Save("x.dat");u.Save("u.dat");t.Save("t.dat");

    plot(xObs1,yObs1,x2,y2,problem.name+": x-y Obstacle coordinate
    trajectory","x", "y", " obstacle trajectory");
    plot(t,u,problem.name+": joints Torque","time (s)","control","u1 u2");
    plot(t,x,problem.name+": links states","time (s)", "states", "q1(rad)
    q1dot(rad/sec) q2(rad) q2dot(rad/sec)");
    plot(t,l,problem.name + ": costates(lambda)", "time (s)", "lambda", "q1
    dq1 q2 dq2");
        }
```

.2 DMOC Code

The Discrete Mechanics for OCP's setup and code for implememting the same problem above with IPOPT and an AMPL interface is specified here. The resulting NLP from the discretization process is also solved for roots using IPOPT.

```
# Number of variables (this is a scalable formulation)
# International Business Machines
# This file is part of the Ipopt open source package, published under the
    Eclipse Public License IBM 2009-04-02
# Author:  BJ
# IPOPT/ AMPL formulation of the coding problem 2 link using DMOC
# Jacobian / gradients/hessian computed using the ADOL-C software.

param A   :=0.344933604;
param B   :=0.018800336;
param C   :=0.007882111306;
param D   :=0.12969075;
param pi  :=3.14159;
param q1r :=1.5;
```

```
param q2r :=1.5;

param as1 :=0.065957;
param bs1 :=0.0088631;
param as2 :=0.030155;
param bs2 :=0.0041179;
param N   ;
param xd  :=0.43;
param yd  :=0.05;
param r1  :=0.16;
param r2  :=0.083545;
param l1  :=0.32;
param l2  :=0.21;
var tf >= 0;
var h  =  tf/N;
#var h = 0.002;



# Definition of the variables with bounds and starting point
var q1 {k in 0..N} :=0; #<=-10, >=10,
var q2 {k in 0..N} :=pi/3; #<=-10, >=10,
var u1 {k in 0..N} ;#: 30 <= u1[k] <= 30;
var u2 {k in 0..N}; #<=-6, >=6;

let N:=24;#(discretization time steps maybe changed.)

#performance index
minimize obj: sum {k in 1..N-1} 0.5*h*((q1[k]-q1r)/h)^2 + sum {k in 1..N-1}
    0.5*h*((q2[k]-q2r)/h)^2 + sum {k in 1..N-1} 0.5*h*u1[k]^2+ sum {k in
    1..N-1} 0.5*h*u2[k]^2;
#sum{k in 1..N} 0.5*((q1[k]-q1r)/h)^2 + 0.5*((q2[k]-q2r)/h)^2
    +0.5*(u1[k]^2+u2[k]^2);

#DELE constraint
```

```
subject to DELE: sum {k in 1..N-1} A*2*((q1[k+1]-q1[k])/h)^2- sum {k in
    1..N-1} B*2/h*((q2[k+1]-q2[k])/h)-sum {k in 0..N-1} D*1/h*(
    ((q2[k+1]-q2[k])/h) + 1/h*((q1[k+1]-q1[k])/h) ) -sum {k in 1..N-1}
    C*(1/h*((q2[k+1]-q2[k])/h)*cos((q2[k+1]+q2[k])/2) +
    1/h*((q1[k+1]-q1[k])/h)*cos((q2[k+1]+q2[k])/2) +
    0.5*((q1[k+1]-q1[k])/h)*((q2[k+1]-q2[k])/h)*sin((q2[k+1]+q2[k])/2)) + sum
    {k in 1..N-1} A*2/h*((q1[k]-q1[k-1])/h)+sum {k in 1..N-1}
    B*2/h*((q2[k]-q2[k-1])/h) + sum {k in 1..N-1} D*1/h*(
    ((q2[k+1]-q2[k])/h)+((q1[k+1]-q1[k])/h) )+ sum {k in 1..N-1}
    C*(1/h*((q2[k+1]-q2[k])/h)*cos((q2[k+1]+q2[k])/2)+1/h*...
((q1[k+1]-q1[k])/h)*cos((q2[k+1]+q2[k])/2)-0.5/h*((q1[k+1]-q1[k])/h)*...
((q2[k+1]-q2[k])/h)*sin((q2[k+1]+q2[k])/2) ) = 0;

subject to ellipse1 {k in 1..N-1}:(((xd-r1*cos(q1[k]))...
*cos(q1[k])-(yd-r1*sin(q1[k]))*sin(q1[k]))^2)/as1+(((yd-(l1*cos(q1[k])+...
r2*cos(q1[k]+q2[k])))*cos(q1[k])-(xd-(l1*sin(q1[k])+...
r2*sin(q1[k]+q2[k])))*sin(q1[k]))^2)/bs1 >= 1;

subject to ellipse2 {k in 1..N-1}:(((xd-r1*cos(q1[k]))...
*cos(q1[k]+q2[k])-(yd-r1*sin(q1[k]))*sin(q1[k]+q2[k]))^2)/...
as2+(((yd-(l1*cos(q1[k])+r2*cos(q1[k]+q2[k])))*cos(q1[k]+q2[k])...
-(xd-(l1*sin(q1[k])+r2*sin(q1[k]+q2[k])))*sin(q1[k]+q2[k]))^2)/bs2 >= 1;

# Boundary conditions for q1
subject to q1_0:
    q1[1] = 0;
subject to q1_f:
    q1[N] = pi/6;

# Boundary conditions for q2
subject to q2_0:
    q2[1] = pi/3;
subject to vtf:
    q2[N] = pi/6;

#Initial Estimates:
let tf := 2.0;
```

```
solve;

display _total_solve_time, tf;

printf {k in 0..N}:
"%10.5f %10.5f %10.5f %10.5f %10.5f \n",tf*k/N, q1[k], q2[k], u1[k], u2[k] >
    dmoc.sol;

#c1[0] = r1*cos(q1[k]);
#c1[1] = r1*sin(q1[k]);
#c2[0] = (l1*cos(q1[k])+r2*cos(q1[k]+q2[k]));
#c2[1] = (l1*sin(q1[k])+r2*sin(q1[k]+q2[k]));
#adouble p1 = q1[k];
#adouble p2 = q1[k]+q2[k];
#path1=(((xd-c1[0])*cos(q1)-(yd-c1[1])*sin(q1))^2)/as1+(((yd-c2[0])...
*cos(p1)-(xd-c2[1])*sin(q1))^2)/bs1 >= 1;
#path2=(((xd-c1[0])*cos(p2)-(yd-c1[1])*sin(p2))^2)/as2+(((yd-c2[0])...
*cos(p2)-(xd-c2[1])*sin(p2))^2)/bs2 >= 1;
```