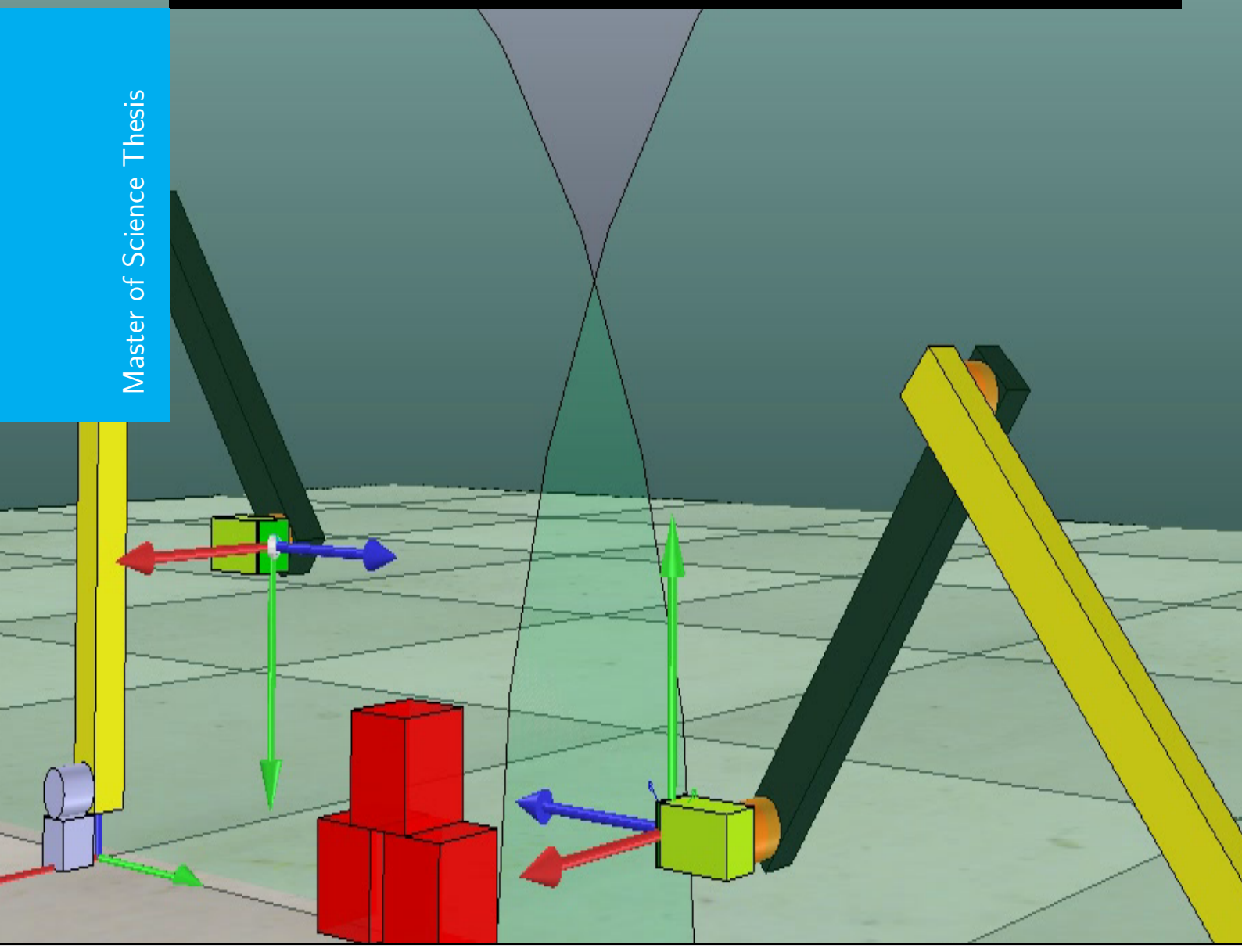
Cooperative Sequential Composition Control for Compliant Manipulation

An Approach via "Robot Contact Language"

Anuj Shah

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Cooperative Sequential Composition Control for Compliant Manipulation

An Approach via "Robot Contact Language"

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Anuj Shah

August 20, 2015

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

A convoluted manipulation task involves extensive planning and the use of a supervisory controller to execute the desired task. One controller specification is generally unsuitable to perform the complete manipulation task. Manipulation involves contact of the object being manipulated with robots, surfaces and other objects in the scenario. The dynamics of a manipulation change when contact amongst these components are made or broken. Using this paradigm of contact amongst robots, objects and surfaces, the Robot Contact Language (RCL) is developed.

Using a combinatory logic, the set of all possible contact combinations of the given components can be generated. Using a few simple rules such as making and breaking contacts, a "contact map" is then devised. On a symbolic level, a potential manipulation task can already be planned with the help of this map by traversing from the initial contact combination or "contact mode" to the goal contact mode. The contact map is enriched with the available geometric information (robot workspaces, surface geometry, etc.) and spatial relationships can then be defined amongst these contact modes for manipulations and mode transitions.

Given the initial and final position of an object to be manipulated, planning begins with a simple graph search on the contact map for the shortest path form the initial to the final contact mode. The modes present in this path automatically divide the task into various subtasks. Manipulation planning can then be done "locally" in each of these contact modes with the aim to proceed further towards the final goal. Each of these sub-tasks can be achieved with a dedicated controller specification. A supervisory controller is needed to bring all these set of controllers together and execute them in a hybrid manner. For this purpose, the idea of sequential composition has been used. By defining the domains of attraction of each controller and the goal-sets already lying in the overlapping state-spaces, the controllers are executed sequentially to achieve the overall manipulation task.

As contact involves interaction forces, a study is also done to understand the working of a spatial spring based impedance controller to achieve compliance in the robotic arm. Simulation have been conducted on Matlab and VREP software to validate the usage and reliability of manipulation planning based on contact maps as well as the performance and versatility of the compliant robotic arm.

Table of Contents

List of Figures

List of Tables

Preface

This book documents my thesis work as an M.Sc. student at Delft University of Technology in the department of Delft Centre for Systems and Control. With my profound interest in the field of robotics, I chose to work in the area of robotic manipulation, collaborating with M.Sc. Esmaeil Najafi, one of my supervisors, in his work on sequential composition, which is a form of hybrid switching control necessary in any robotic system.

Along the course of work, me along with my supervisors Dr. Gabriel Lopes and Esmaeil came up with the idea of developing a *contact language* which would help define a complex robotic manipulation task at a higher level, easily understood by humans and simple enough for a robot task planner to decipher the task to be carried out. Most of my later work involved developing this contact language - the nomenclature, design, usage etc.

Using this newly developed language, various simulations were performed which carried out task planning, control design and their execution in a hybrid fashion using the ideas of sequential composition as well as force controlled and compliant robotic manipulators. The modelling, planning and control design were carried out in MATLAB. The execution was done in a software called Visual Robotics Experimentation Platform (VREP) using a couple of self-designed planar robotic arms.

The book consists of six chapters, each addressing a separate topic. Chapter 1 gives an introduction to all the subsequent chapters. The main matter starts from Chapter 2 which deals with the modelling and control of compliant robotic arms. The focus is mainly on compliance control and cooperative manipulation. It also introduces the idea of sequential composition along with a detailed theory and its applications. This is followed by Chapter 3 which introduces the concept of *Robot Contact Language (RCL)* along with its use in robotic systems.

Chapter 4 brings the worlds of robotic manipulation, sequential compositions and Robot Contact Language together and describes a complete control synthesis for a complex robotic manipulation task followed by Chapter 5 which documents various simulation results of the complete controller synthesis and task execution. The final chapter concludes and summarizes the work carried out in the thesis and suggests a few ideas concerning future work.

I would like to thank my parents for all the help and support they have provided me throughout my life and without whom I would have not had this incredible opportunity to move away from home and explore this wonderful and amazing world.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, dr. G.A.D. Lopes for the opportunity, guidance and help throughout my M.Sc. thesis. I would also like to thank E. Najafi for giving me an opportunity to collaborate with his Ph.D. work and guiding me throughout the thesis.

I am also thankful to all the committee members for taking their valuable time out for my thesis assessment.

Delft, University of Technology — Anuj Shah
August 20, 2015

"They took our jobs!!"
— *South Park*

Chapter 1

Introduction

The use of robotic manipulators has come a long way, from the very early, remote controlled robots to automated manipulators in the industrial environment for tasks like welding, painting, assembly, etc. to the current humanoid robots which operate in non-isolated and cluttered environments to perform complex tasks like folding laundry, serving beverages and such similar tasks, which require higher level decision making, along with sophisticated hardware and software integration and intelligent control algorithms.

A complex and convoluted manipulation task involves decision-making at various stages of planning and execution of the manipulation. A single controller generally does not suffice to carry out a complex manipulation task. Take for example, the task where a humanoid robot (with two robotics arms) is required to pick and place some glassware from a shelf on to a dining table. This task requires planning and control at several stages. Clearly, the task involves contact of the robotic arm with various objects being manipulated. The planning itself is not straightforward. The robot has to first grasp the object safely, manipulate it while avoiding obstacles and then place it on the correct surface. It needs to perform several manipulations in a correct sequence to achieve the final goal. If the object being manipulated is larger, perhaps a wide serving tray, the robot will have to use both its arms to manipulate one single object, that is, perform a bimanual task. If it has to pick and place several different objects which are co-dependent on each other, the robot also has to decide the sequence of object manipulation in order to successfully accomplish the entire task.

On having described a few of the important aspects of the aforementioned manipulation task, there are several research areas and problems that need to be analysed. A few of them have been identified and studied in this research work which are explained in the following subsections followed by a detailed analysis in the subsequent chapters.

1-1 Robotic Manipulation

Robotic manipulation can be defined as the control of a robot in a skilful manner to accomplish a desired task. There are several aspects to robotic manipulation, as explained in the following.

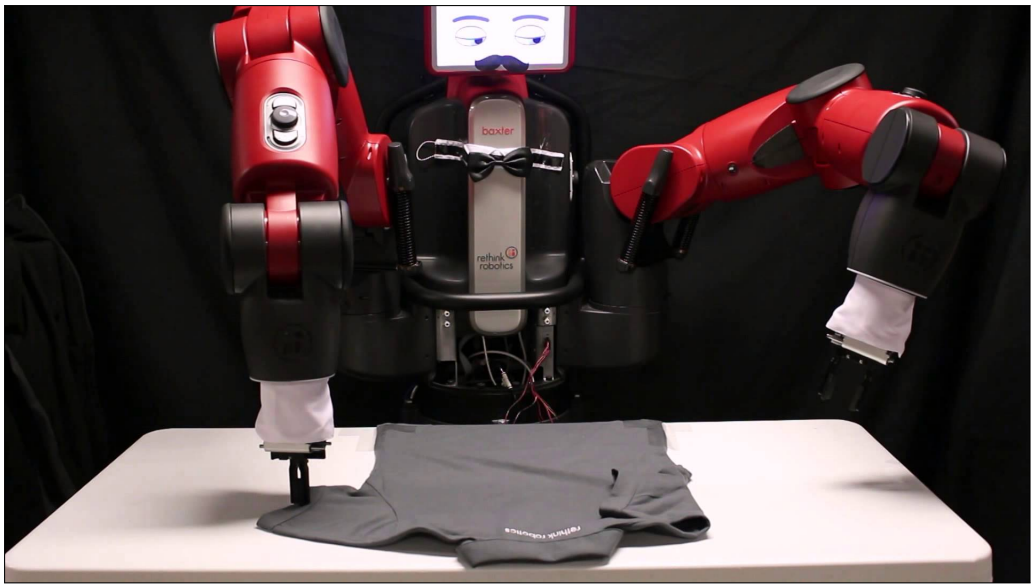


Figure 1-1: Baxter robot folding a t-shirt. Image taken from Rethink Robotics Youtube channel [2].

1-1-1 Modelling and Control

In order to control a robot or more specifically, a robotic arm, we require a model of the system for accurate and precise control. There are generally two types of models available for a robotic system - the kinematic model and the dynamic model [3, 4]. As the names suggest, the kinematic model only describes the kinematics (motion) of the robotic arm. On the other hand, a dynamical model describes the dynamics of the arm which involves masses and forces along with motion variables. Both these models could be used for controlling the arm.

The control of a robotic arm involves positioning of the robot joints or end-effector in a certain configuration or maintaining certain force at the end-effector or both. Position control is considered to be a trivial task as compared to controlling the forces at the end-effector. The end-effector force control is a special class of control problem, commonly referred to as *compliance control* [5, 6].

1-1-2 Compliance Control

The tasks performed by robotic manipulators are broadly divided into two categories, viz. contact and non-contact tasks [7]. While a non-contact task only involves controlling the robotic arm in a certain position and orientation, for contact tasks a robotic arm needs to have a control on the forces exerted by itself on the environment. In formal terms, the robotic arms need to be force controlled. Non-contact tasks are unconstrained in nature, for example, spray-painting, glueing, etc. In contrast, a contact based tasks involves interaction of the robot with the environment and sometimes requires simultaneous control of its position and exerted forces. Examples of such tasks are grinding, bending, assembling, and many more.

Compliance can be considered as the ability of the robotic arm to react to the interaction forces (between the robot and the environment). Compliance control techniques can be broadly

divided into two categories [8]:

- **Passive compliance:** compliance is achieved due to hardware or structural modification.
- **Active compliance:** compliance is achieved via software using force feedback or other possible control methods.

Active compliance has a huge advantage over passive as it can be easily modified by changing a piece of software compared to structural modifications needed to change the properties of passive compliance. In modern robotic platforms, passive compliance is used along with active compliance to achieve good compliance properties. In this research work, the author's focus is mainly on active compliance. This can be achieved through two main categories of control techniques:

- **Hybrid position/force control:** both, the desired force and the position are controlled in orthogonal spaces depending on the task [5, 6].
- **Impedance control:** the robotic arm behaves as a (virtual) spring and the force is proportional the desired and actual position of the arm [9, 10].

Both the above stated categories of active compliance control techniques were explored. The technique used in the simulations is based on an impedance control technique and hence, for brevity, only this particular control technique will be covered in detail.

1-1-3 Cooperative Manipulation

Cooperative (robotic arm) manipulation mainly refers to manipulation of an object with two or more robotic arms. This domain seems like a trivial and straightforward application of single arm manipulation extended to multiple arms. However, it poses several challenges as the multiple arms integrate into one big system and several (kinematic and force) constraints have to be satisfied simultaneously for a successful manipulation [22, 8, 1].

The focus of the author's work is mainly on dual-arm systems due to ease of understanding and its intuitive nature, due to similarities with the human anatomy. The ideas and techniques can easily be extrapolated to systems with more than two arms.

There are several reasons for studying dual-arm systems some of which are as follows [7]:

- Transferring skills: similarity to the operator in tele-manipulation operations make them ideal for using them to perform bimanual tasks and transfer the operators' skills required by the task.
- Flexibility and stiffness: by using two arms in closed kinematic chain the strength of a parallel manipulator and the flexibility of a serial chain can be achieved simultaneously.
- Manipulability: by using multiple arms, two parts of a task can be controlled and tasks which cannot be performed by a single arm can be performed effectively.

- Interaction with the environment: most of the current robots are restricted to industrial setups, and are unsafe for use in an unstructured environment and in the presence of living beings. Research is vital in humanoid robotics as a replacement for performing human-like bi-manual tasks in such unstructured environments.

Many such motivating reasons have led to the development of a large number of dual arm platforms, like DLR's Rollin Justin [11], NASA's Robonaut [12], Rethink Robotics' Baxter [13] and many more, but there are still various common challenges that need to be addressed. Cooperative manipulation requires the need of simultaneous position and force control for proper object grasping and manipulation and as a result, the robots need some form of compliance to control its forces. To generate the paths for various robots involved in a manipulation, there is also a need to generate the path of the object being manipulated first, along with geometrical information of the object for proper grasping [1].

The research presented here is mainly concerned with bimanual tasks, where there is a physical interaction of both the arms with the same object. Applications of cooperative manipulation range from domestic to industrial domain [7]. One of the most studied and researched bimanual domestic application is of folding laundry. Attendant care and kitchen support for the elderly is another application which is being currently researched. In the industrial domain, cooperative bimanual tasks mainly involve pick and place and assembly of larger objects.

1-2 Supervisory Control

Picking and placing glassware from a shelf onto a table is a complex task. As mentioned earlier, it involves several different manipulations like approaching the object, grasping the object and leaving contact with the object. In such a situation, it is evident that one control specification will not be able to accomplish all these tasks as they all require different conditions to be satisfied.

This leads to an idea of breaking up a complex task into several sub-tasks and assign specific controllers that (only) carry out these sub-tasks. If one mixes or *composes* these controllers in an intelligent way and executes them in the right *sequence*, the given complex task can be achieved with ease. This is the basic idea of *sequential composition control (SCC)* [14, 15], which acts as a supervisory controller to this pool of controllers.

SCC is a control system framework that applies a set of local controllers sequentially to reach a goal that can not be achieved by a single controller. It is not possible in every controlled system to reach a desired state using just one controller, nor is it necessary that the *goal-set* is in the *domain of attraction (DoA)* of that controller. In such a case, a number of controllers can be applied sequentially, with overlapping DoAs, reaching the DoA of the controller that can bring the system to the desired final state, and executing the final controller to actually reach the desired final states. Sequential compositions have been successfully used with many systems where there are obstacles between the desired and initial position or one controller simply cannot perform the desired task. By carefully composing various local controllers, the desired goal can be achieved [14, 16].

1-3 Robot Contact Language

It can be observed that robotic manipulation involves contact of the object being manipulated with robots, various surfaces and even other objects. When a robot is commanded to place the glassware from the shelf onto the table, it could be - "Pick up the glass from the shelf and put it on the dining table." There are several keywords in this statement that would translate to the object being in contact with the shelf initially and needs to be in contact with the dining table (at some specific location). The manipulation task of grasping the glass from the shelf and placing it onto the table would involve the contact of a robot arm with the glass.

This lead us to the idea of developing a higher level "robot language" which represents the configuration of the object (being manipulated), at any instance of time, in terms of contact with one or more components, be it robots, static surfaces or other objects. This is the basic idea behind *Robot Contact Language (RCL)*. It has been seen that for a given complex task, a need arises to break down the task into several different sub-tasks and execute them using different controllers. Representing the manipulation task in RCL already does that to a great extent.

In a given complex manipulation task, the robot has to pass through various contact phases or *contact modes* to reach its final contact mode and consequently, its goal position. Using spatial relationships amongst the workspaces of robots, surfaces and objects, a *contact map* can be generated which involves the combinations of all contact modes in a given manipulations along with the transition to other contact states based on these spatial relationships.

Given the initial and final position of an object to be manipulated, the feasibility check of the given task is just a matter of finding a path in the generated contact map from the initial to the final contact mode. If feasible, the execution of the task is a straightforward assignment of manipulation controllers for each of these contact states and a transition controller between them (if needed). Using the idea of SCC, these controllers can be sequentially composed and executed in a hybrid fashion to achieve the task.

Using RCL and contact maps, a manipulation task can thus be carried out using a few easy step which include the generation of the robot contact modes and contact map, finding the shortest path on this map for the given manipulation, path-planning and controller assignment for each relevant contact mode and execution using sequential compositions.

1-4 Summary

This chapter served as an introduction to the research topics dealt by the author. Starting with an example of a complex manipulation to provide an easier way of visualization for the readers, it gave an introduction to robotic manipulation and its primary aspects. The need of compliance in robotic arms was explained followed by a short introduction to cooperative manipulation. This chapter also introduced the idea of sequential composition control and its importance. It concluded with a brief introduction to a new concept of Robot Contact Language and contact maps which are used to simplify the planning of complex manipulation task.

Chapter 2

Theoretical Background

2-1 Introduction

This chapter talks about the basic theoretical and mathematical background of robotic manipulation and sequential composition control (SCC). It gives a basic description of robotic manipulator and the mathematical representation of the various terms related to it, followed by a basic modelling and position control. As force or compliance control is studied in much detail in this research, the compliance control method used in the simulations has been explained in more detail.

The second part of this chapter describes the basics of SCC. The theory is followed by a simple example and motivation to study sequential compositions. The reader is advised to look up the citations for a descriptive mathematical background.

2-2 Robotic Manipulation

A robotic manipulator can be considered to be set of rigid links connected via joints which cause their movement. The joints of a robotic arm can be revolute or prismatic in nature. Motors attached to each of these joints cause these joints, and consequently, the robot links to move in space. These joints undoubtedly constrain the movement of the links to a certain axis. These links can be attached in series, like a chain, or a set of rigid links can be attached to a common joint from one end. The former type of robotic manipulator is know as an *open chain* manipulator and the latter is called a *parallel* manipulator. In this research study, the focus is only on open chain manipulators.

The joint angles/positions are commonly know as *generalized coordinates* or *configuration coordinates*. Considering an n-link manipulator, the number joints would also be equal to n. Let vector $\boldsymbol{q} = (q_1, q_2, ...q_n)^T$ represent the set of all joint positions, with $\boldsymbol{q} \in Q$, where Q is the *joint space* of the robot.

The *end-effector* of the robot is basically the end-point of the robot's n^{th} link, where the manipulation tool (grasper, welder, sprayer, etc.) is attached. As the primary interest is in controlling the robot's end-effector, a representation of its position, velocity and the force vectors is needed.

It is customary to represent the positions of robot link (or any component in the scene[1]) with respect to a common, *base frame* $\boldsymbol{\Psi_0}$. The base frame is generally chosen as the base of the robot, but it can be chosen to be anywhere in the *Cartesian space*. After choosing an inertial frame of reference, a coordinate frame can be attached to any component in the scene. As we are interested in the position and orientation of the robot's end-effector, a frame $\boldsymbol{\Psi_n}$ is attached to the end of the n^{th} link (i.e., at the end-effector) and is called the *end-effector frame*. This frame is represented by a *homogeneous matrix* $\boldsymbol{H} \in \mathbb{R}^3 \times SO(3)$. This homogeneous matrix is of the form

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{p} \\ \boldsymbol{0} & 1 \end{pmatrix}, \tag{2-1}$$

where $\boldsymbol{R} \in SO(3)$ is a 3×3 *rotation matrix* giving the orientation of the frame and $\boldsymbol{p} \in \mathbb{R}^3$ is the *translational position vector* representing the position of the frame, about and along the x, y and z axes, respectively.

The velocity vector of a frame with respect to an inertial frame is given by the *twist* vector $\boldsymbol{T}$. Twists are basically velocities of rigid bodies in $SE(3)$. Geometrically, they are elements of Lie algebra $se(3)$ associated to the $SE(3)$ Lie group. Twists are in the following form:

$$\boldsymbol{T} = \begin{pmatrix} \boldsymbol{w} \\ \boldsymbol{v} \end{pmatrix}, \tag{2-2}$$

with $\boldsymbol{T} \in \mathbb{R}^6$. Here $\boldsymbol{w}$ is the *angular velocity* of the frame with its skew-symmetric form $\tilde{\boldsymbol{w}}$ belonging to $so(3) := \mathbb{R}^{3\times3}$ and $\boldsymbol{v} \in \mathbb{R}^3$ is the *translational velocity*, again, about and along the x, y and z axes, respectively.

A force can similarly be represented by a co-vector and is commonly referred to as *wrench*. Wrenches are duals of twists and can be considered to be linear operator from twists to Power. Similar to twists, wrenches can be written as a 6D co-vector of the form:

$$\boldsymbol{W} = \begin{pmatrix} \boldsymbol{m} & \boldsymbol{f} \end{pmatrix}, \tag{2-3}$$

with $\boldsymbol{W} \in \mathbb{R}^6$. Here, $\boldsymbol{m}$ is the 3D *torque co-vector* and $\boldsymbol{f}$ is the 3D *force co-vector*. More details about twists and wrenches, their coordinate transformation and their other important properties can be found in [3]. The author assumes that the reader is familiar with these details from this point onwards. The next section covers the kinematic and dynamic modelling of an open chain robotic manipulator system.

[1] A manipulation "scene" refers to the environment where manipulation is taking place. A scene includes all the robots, objects and surfaces present in the and around the environment relevant in the task.

2-3 Modelling of Robotic Manipulators

For precise control of a robotic arm, a model of the system is generally needed. The main parameters that need to be controlled are the position, orientation, twists and wrenches of the end-effector. In robotic systems, one can work with two types of models - *kinematic model* and *dynamic model*. A kinematic model only relates the motion parameters of the joint positions with the end-effector. A dynamic model relates the inertia terms and other forces acting on the robot as well.

The position of the end-effector with respect the all the joint positions, also know as the *direct kinematics* or *forward kinematics*, is given by the famous *Brokett's Exponential Formula*, which is,

$$\boldsymbol{H}_n^0(\boldsymbol{q}) = e^{\tilde{\hat{\boldsymbol{T}}}_1^{0,0} q_1} e^{\tilde{\hat{\boldsymbol{T}}}_2^{0,1} q_2} \dots e^{\tilde{\hat{\boldsymbol{T}}}_n^{0,n-1} q_n} \boldsymbol{H}_n^0(0). \tag{2-4}$$

In this equation, the end-effector has the n^{th} frame attached to it and the position and orientation is given in the form of a Homogeneous matrix $\boldsymbol{H}_n^0(\boldsymbol{q})$ as a function of the joint positions $\boldsymbol{q}$. $\tilde{\hat{\boldsymbol{T}}}_n^{0,n-1}$ represents the skew-symmetric form of the unit twist vector of the n^{th} frame, with respect to the $(n-1)^{\text{th}}$ frame in the base or zero frame. $\boldsymbol{H}_n^0(0)$ represents the initial position of the end-effector when all joint positions are zero, i.e., $\boldsymbol{q} = \boldsymbol{0}$. Equation (2-4) is a non-linear map from joint positions to end-effector position and orientation.

The joint velocities, on the other hand, transform via a linear map to give the end-effector twist. This transformation matrix is commonly know the *Jacobian matrix* and is dependent on the joint positions. The relationship is given as:

$$\boldsymbol{T}_n^{0,0} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{2-5}$$

$\boldsymbol{T}_n^{0,0}$ is the end-effector twist with respect to the base frame in base frame. $\dot{\boldsymbol{q}}$ is the joint velocity vector which transforms via the Jacobian matrix $\boldsymbol{J}(\boldsymbol{q})$. (2-4) and (2-5) together give the kinematic model of a given robotic system. The detailed derivation and analysis can be found in [3, 4].

The dynamic model of a robotic manipulator, as the name suggests, describes the dynamics of the robotic system. A general form of a manipulator dynamical equation (without friction and other losses) is given as:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau}^T + \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{W}_{\text{ext}}^T. \tag{2-6}$$

This is a non-linear set of differential equations which describes the dynamics of a robot. $\boldsymbol{M}(\boldsymbol{q})$ is the manipulator inertia matrix, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is called the Coriolis matrix and $\boldsymbol{G}(\boldsymbol{q})$ is the gravity compensation matrix. $\boldsymbol{\tau}$ is the joint torque co-vector and $\boldsymbol{W}_{\text{ext}}$ is the external force acting at the robot end-effector, which transforms to the joints via the transpose of Jacobian matrix. With some basic transformations, this equation can equivalently be written in Cartesian space in terms of end-effector positions, twists and wrenches [11].

The dynamical equation is generally used for precise robot control where the non-linear motions of the robot are not wanted. The next section briefly describes basic robot control.

2-4 Control of Robotic Manipulators

A robotic manipulator can be controlled directly in joint space or in Cartesian space. The control problem is either *regulation* or *tracking* of a desired set-point or trajectory respectively. This comes under the category of *position control*. In joint space, the desired positions and velocities are that of the various joint positions. In Cartesian space control, the end-effector is required to regulate or track the desired end-effector frame. Cartesian space control is equivalently know as *workspace control* as it takes place in the working space of the robot.

When the robot is in contact with an external environment (in most cases, via its end-effector), its motion is constrained in certain directions. It is desired that the robot end-effector applies a certain amount of force to the external environment or the forces should not exceed a certain threshold; the interaction forces need to be controlled. This requires manipulator *force control* and the robot to be less *stiff* or more *compliant* during interaction.

The next part covers the basic position control methods in joint space and in workspace. These techniques are found vastly in the literature and thus will be dealt with in brief. The section following that will discuss compliance control along with a detailed description of one the impedance control [9] based compliance control methods used by the author in the research work.

Position control techniques for robotic manipulators range from basic PID based control to model-based computed torque control. Control is directly in joint space or in the robot workspace. Most of the workspace control techniques require the use of *inverse kinematics (IK)*. The following enumerates the important categories of position control techniques, which can equivalently be applied in joint space and also in workspace via IK methods [4, 8].

- **P(I)D control:** Refers to the most commonly used control technique in the field of control engineering. Proportional and derivative gains are chosen to track or regulate the desired position and velocity in a critically damped manner. Gravity compensation is added for more accurate control. The robot behaviour is non-linear in both joint space and the workspace. The control law (in joint space) can be given as:

$$\boldsymbol{\tau}^T = \boldsymbol{K}_\mathrm{d}(\dot{\boldsymbol{q}}_\mathrm{d} - \dot{\boldsymbol{q}}) + \boldsymbol{K}_\mathrm{p}(\boldsymbol{q}_\mathrm{d} - \boldsymbol{q}) + \boldsymbol{G}(\boldsymbol{q}) \tag{2-7}$$

  where, $\boldsymbol{\tau}$ is the input joint torque co-vector, $\boldsymbol{K}_\mathrm{d}$ and $\boldsymbol{K}_\mathrm{p}$ are derivative and proportional gains respectively and $\boldsymbol{G}(\boldsymbol{q})$ is the gravity compensation matrix. An integral control term can be added to further reduce steady-state errors.

- **Computed torque control:** This is a model based control which cancels out all the system non-linearities and renders the robot linear (in joint space or workspace, depending on the method). The control method, most certainly, requires an accurate robot model as given in (2-6) and also increases the computations. This is because the model is dependent on joint positions and velocities and hence, the computations have to be carried out in each step. The control input is given by:

$$\boldsymbol{\tau}^T = \boldsymbol{M}(\boldsymbol{q})(\ddot{\boldsymbol{q}}_\mathrm{d} + \boldsymbol{K}_\mathrm{d}(\dot{\boldsymbol{q}}_\mathrm{d} - \dot{\boldsymbol{q}}) + \boldsymbol{K}_\mathrm{p}(\boldsymbol{q}_\mathrm{d} - \boldsymbol{q})) + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}). \tag{2-8}$$

  It can be seen that the model matrices cancel out the system non-linearities and the robotic arm is linear (in joint space).

2-5 Compliance Control

In a manipulation task where there is interaction of the robot with the an external environment, perhaps an object or other robots, interaction forces are present. These forces acting on the external environment need to be controlled to avoid damage to the robot or the environment or perhaps, to carry out the task with some desired interaction forces. In such cases, pure position control will inevitably fail. The motion of the robot's end-effector in such cases is either constrained in certain directions, thus imposing the so called *kinematic constraints* or the contact task might be characterized by dynamical interaction between the robot and the environment, for example, a robot pushing a box. Whatever be the case, some form of force or compliance control is necessary to carry out such tasks.

The motivation behind developing compliance control can be explained by a simple example of robot inserting a peg-in-the-hole as shown in Figure 2-1 [5, 8]. As long as the planning is perfect, the peg insertion can be achieved via pure position control. A small position error in the xy-plane though, will lead to very high interaction forces and cause damage to the robot and the objects. If on the other hand, the robot has some compliance, the peg insertion would take place easily without high interaction forces.

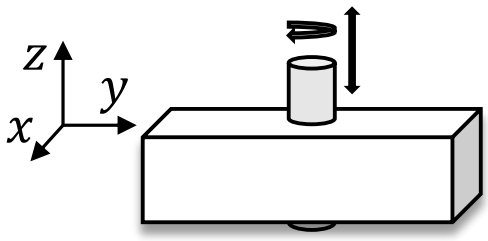


Figure 2-1: A schematic of peg-in-the-hole task.

Compliance, by definition, is the property of a material to undergo elastic deformation when subjected to external force. The term compliance control is broadly and generically used in the literature to describe a control strategy which involves explicit or implicit control of interaction forces acting on the robot or its end-effector [8].

The compliance control strategy studied and used in the author's research is based on the famous impedance control [9, 10] as described in the following.

Impedance control is one of the most often used compliance control techniques. The name comes from the electrical engineering term "impedance" which is to impede or resist the flow of electric current. Analogically, it can be used for the interaction of rigid bodies in motion. This control strategy makes the end-effector of a robotic manipulator behave like a mass-spring-damper like system. By varying the stiffness of this "spring" (and other parameters), the desired compliance (or stiffness) of the end-effector can be achieved depending on the task it is performing.

To understand the concept of impedance control, we can consider a one degree-of-freedom spring. Let the rest length of the spring be x_{d} with respect to an inertial frame and let x be its current position with respect to the same inertial frame. The force acting on the spring F_{spg} is proportional to the different between its the current and rest length, that is,

$$F_{\text{spg}} = K_{\text{spg}}(x - x_{\text{d}}) \tag{2-9}$$

with the stiffness constant being K_{spg}. This same idea can be extended to a six-dimensional [8] case for a robotic manipulator and the impedance behaviour of the end-effector can seen in the form:

$$\boldsymbol{\Lambda}_{\text{d}}\ddot{\tilde{\boldsymbol{x}}} + \boldsymbol{D}_{\text{d}}\dot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_{\text{d}}\tilde{\boldsymbol{x}} = \boldsymbol{W}_{\text{ee}}^{T} \tag{2-10}$$

where $\boldsymbol{\Lambda}_{\text{d}}$, $\boldsymbol{D}_{\text{d}}$ and $\boldsymbol{K}_{\text{d}}$ are the desired end-effector inertia, damping and stiffness matrices, $\boldsymbol{W}_{\text{ee}}$ is the generated end-effector wrench and $\tilde{\boldsymbol{x}}$ is an infinitesimal twist of the end-effector given as $\tilde{\boldsymbol{x}} \sim \delta\boldsymbol{T} = [(\delta\boldsymbol{\theta})^T(\delta\boldsymbol{p})^T]$. Once the end-effector wrench has been calculated, it can be translated via transpose Jacobian to generate joint torques for control.

This complete form of impedance control requires the computation of inverse Jacobian and robot model [11, 8]. Thus, many practical implementations of impedance control avoid inertia shaping in order to avoid IK. One of the implementations of impedance control uses only the kinematics of the robot to make the robot behave like a spatial or virtual spring. Thus, no dynamical model is required. This implementation has been described in some detail. Further reading can be found in [17, 18].

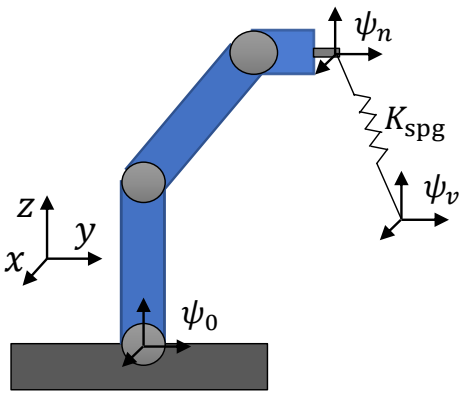


Figure 2-2: A schematic of robotic manipulator control via spatial spring. Here, $\boldsymbol{\Psi_n}$, $\boldsymbol{\Psi_v}$ and $\boldsymbol{\Psi_0}$ are the end-effector, virtual and base frames respectively and K_{spg} is the spatial spring constant.

A *spatial spring* or a *virtual spring* is a method of defining and generating a behaviour in a robotic system, similar to an actual mechanical spring. For a robotic manipulator, this spring is defined between the actual end-effector position and a desired or *virtual position*. This virtual position is also called a *virtual object*. A frame is attached to both, the actual and virtual positions of the end-effector. The virtual frame $\boldsymbol{\Psi_v}$ will vary depending on the task. This spring with varying stiffness parameters between the two frames gives compliance to the robot.

If there are no external forces acting on the robot, both the frames are aligned. If a force is acting on the robot, the force at the end-effector is proportional the distance between the two frame (in $SE(3)$). An energy function can be defined between these two frames, similar to the energy in a one dimensional mechanical spring, given as,

$$E_{\text{spg}} = \frac{1}{2} K_{\text{spg}} (x - x_{\text{d}})^2 \tag{2-11}$$

but, in $SE(3)$. It is trivial to see that the minimum energy of the system will be at the equilibrium point, where both, the actual and virtual end-effector frames coincide. To stabilize the end-effector frame at the virtual frame, a virtual damping term is added to dissipate free energy (similar to that from equation (2-10)).

On setting the two frames, the stiffness matrices for the task are chosen. These correspond to translational stiffness, $\boldsymbol{K}_{\text{t}}$, rotational stiffness $\boldsymbol{K}_{\text{r}}$ and coupling stiffness $\boldsymbol{K}_{\text{c}}$. The coupling stiffness relates how much a displacement in the translational axes will affect the rotational axes and vice-versa. The desired stiffness matrix is given as:

$$\boldsymbol{K_{\text{d}}} = \begin{bmatrix} \boldsymbol{K_{\text{r}}} & \boldsymbol{K_{\text{c}}} \\ (\boldsymbol{K_{\text{c}}})^T & \boldsymbol{K_{\text{t}}} \end{bmatrix}. \tag{2-12}$$

With some mathematical manipulation, these matrices are converted in to the corresponding co-stiffness matrices $\boldsymbol{G}_{\text{t}}$, $\boldsymbol{G}_{\text{r}}$ and $\boldsymbol{G}_{\text{c}}$ which make the computations easier.

The actual end-effector frame can be defined in terms of a homogeneous matrix as $\boldsymbol{H}_n^0$ and the virtual position frame as $\boldsymbol{H}_v^0$. As seen, both the frames are defined with respect to a common base frame. The relative position between these two frames can thus given as:

$$\boldsymbol{H}_n^v = \boldsymbol{H}_0^v \boldsymbol{H}_n^0 = (\boldsymbol{H}_v^0)^{-1} \boldsymbol{H}_n^0. \tag{2-13}$$

Using the co-stiffness matrices and the relative positions from Equation (2-13), potential energy functions can be defined for the separate translational, rotational and coupling parts, similar to Equation (2-11). the end-effector wrench, $\boldsymbol{W}^{n,n} = [(\boldsymbol{m}^{n,n})^T (\boldsymbol{f}^{n,n})^T]$ corresponding to these potential energies can thus be computed using the following equations:

$$\boldsymbol{m}^{n,n} = -2 \text{ as}(\boldsymbol{G}_r \boldsymbol{R}_v^n) - \text{as}(\boldsymbol{G}_t \boldsymbol{R}_v^n \tilde{\boldsymbol{p}}_n^v \tilde{\boldsymbol{p}}_n^v \boldsymbol{R}_n^v) - 2 \text{ as}(\boldsymbol{G}_c \tilde{\boldsymbol{p}}_n^v \boldsymbol{R}_n^v) \tag{2-14}$$

$$\boldsymbol{f}^{n,n} = -\boldsymbol{R}_v^n \text{ as}(\boldsymbol{G}_t \tilde{\boldsymbol{p}}_n^v) \boldsymbol{R}_n^v - \text{as}(\boldsymbol{G}_t \boldsymbol{R}_v^n \tilde{\boldsymbol{p}}_n^v \boldsymbol{R}_n^v) - 2 \text{ as}(\boldsymbol{G}_c \boldsymbol{R}_n^v). \tag{2-15}$$

Here, as() represents the anti-symmetric[2] matrix function and $\tilde{\boldsymbol{p}}_n^v$ represents the skew-symmetric form of the relative position vector $\boldsymbol{p}_n^v$. $\boldsymbol{W}^{n,n}$ is expressed with respect to the end-effector frame, but all other values are represented in the base frame or the robot's inertial frame. Coordinate conversion of $\boldsymbol{W}^{n,n}$ is thus done as:

$$\boldsymbol{W}^{0,n} = (\text{Ad}(\boldsymbol{H}_0^n)^T (\boldsymbol{W}^{n,n})^T)^T \tag{2-16}$$

[2]For an arbitrary matrix $\boldsymbol{Q}$, $\text{as}(\boldsymbol{Q}) = \text{tr}(Q)\boldsymbol{I} - \boldsymbol{Q}$, where tr() is the trace of the matrix.

where $\text{Ad}()^T$ represents the Adjoint[3] transpose matrix function and $\boldsymbol{H}_0^n$ is the homogeneous matrix of the base frame with respect to the end-effector frame. A damping component is added to dissipate energy from the spring and the final end-effector wrench is given as:

$$\boldsymbol{W}_{\mathbf{f}}^{0,n} = \boldsymbol{W}^{0,n} - (\boldsymbol{D}_{\mathbf{c}}\boldsymbol{T}_n^{0,0})^T \tag{2-18}$$

where, $\boldsymbol{D}_{\mathbf{c}}$ is the Cartesian damping matrix and $\boldsymbol{T}_n^{0,0}$ is the end-effector twist. The last step is to convert this wrench into joint torques, which is done via the transpose Jacobian. Along with this, A gravity compensation term is also added to counter the effects of gravity. The final joint torque co-vector is given as:

$$\boldsymbol{\tau} = \boldsymbol{J}^T(\boldsymbol{q})(\boldsymbol{W}_{\mathbf{f}}^{0,n})^T + (\boldsymbol{G}(\boldsymbol{q}))^T. \tag{2-19}$$

Now that the spatial spring controller has been defined, the question is, how is compliance achieved and how can it be varied. The answer to this question is as simple as understanding a simple mechanical spring. A stiff spring naturally gives less compliance and vice-versa. Using the spatial spring controller, we have advantage of varying this stiffness by changing the values of stiffness matrices $\boldsymbol{K}_{\text{t}}$, $\boldsymbol{K}_{\text{r}}$ and $\boldsymbol{K}_{\text{c}}$ to achieve the appropriate compliance as per the task being performed. These values can also be manipulated in run time. When the motion of the end-effector is constrained by some external object, the force acting on it is proportional to the relative distance between the virtual and actual end-effector position and the value can be varied by manipulating the values of stiffness matrices and/or changing the virtual position. This way, one can achieve simultaneous position and force control. It is important to note that lesser the stiffness, lesser is the position tracking accuracy of the end-effector (but more the compliance). Thus, there is always a trade-off between force control and position accuracy.

The pros and cons of this compliance controller is listed as follows:

- **Advantages:**
  - simultaneous position and force control
  - dynamical model not required, only requires robot kinematics
  - workspace control without inverse kinematics
  - behaviour of the end-effector close to spring-like
  - robot not unstable when desired position at singularities or outside robot workspace
  - not computationally intensive
  - does not require force (sensor) readings for force control
- **Disadvantages:**

[3]The Adjoint of a Homogeneous matrix is used for coordinates transformations of twists and wrenches. Twists transform via the Adjoint of the homogeneous matrix whereas wrenches transform via transpose Adjoint. The Adjoint of1 a homogeneous matrix $\boldsymbol{H}_i^j$ is given by:

$$Ad_{\boldsymbol{H}_i^j} = \begin{bmatrix} \boldsymbol{R}_i^j & \boldsymbol{0} \\ \tilde{\boldsymbol{p}}_i^j \boldsymbol{R}_i^j & \boldsymbol{R}_i^j \end{bmatrix} \tag{2-17}$$

.

- no inertial shaping
- choosing stiffness and damping parameter not trivial and task dependent
- indirect force control

The advantages of using this form of compliance control overpower the disadvantages. It is ideal for use in humanoid robot applications where the position and force accuracies are not very strict. It would most definitely not work very well in control applications which require the highest precision like chip fabrication. In the author's research, the focus was not on high precision applications and thus, this form of controller was used for further research. The simulation results for this type of controller can be found in the penultimate chapter.

2-6 Cooperative Manipulation

Cooperative manipulation refers to manipulating objects using two or more robotic arms and have been a subject of much study in recent years. Started out mainly for manipulating large objects in the industrial environments, cooperative manipulation has made much advancement in the field of humanoid robotics for manipulating objects in a domestic environment (apart from industrial). There are various reasons and situations where cooperative manipulation is more useful or necessary. Some tasks may require multiple arms due to larger size and/or weight of the object being manipulated. Grasping or geometric constraints may favour cooperative manipulation more than single arm manipulation is some cases. For example, a serving tray carried using two arms is more stable than carried by a single arm.

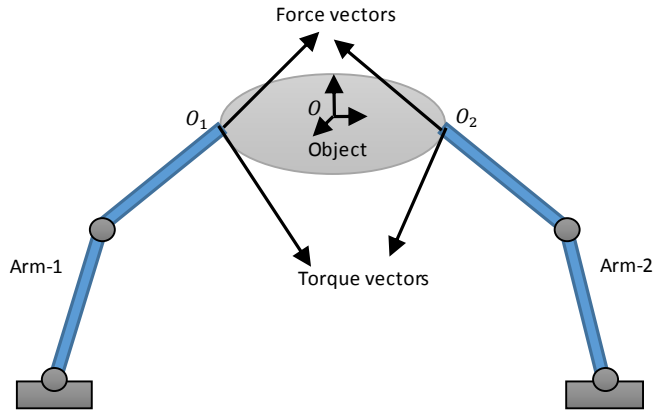


Figure 2-3: A schematic of dual-arm cooperative robotic manipulation of an object.

It is important to point out that cooperative manipulation does require some form of force control to maintain the grasping force to keep the object being manipulated from losing the contact force and slipping. It is also required by the robotic arms to keep the grasping forces from being too high to avoid damage to the object or the robots. Such forces do not cause object motion but lead to internal stress and are generally referred to as *internal forces* [7, 1] in cooperative manipulation literature. Grasping points should be chosen with careful attention to the object geometry. If the centre of mass does not lie on the axis of the line joint the

two (or more) grasping points, there may be unwanted moments that might cause unwanted motions of the objects. These are called *external forces*. These internal and external forces can (and have to) be implicitly or explicitly controlled via various cooperative manipulation techniques, which are generally based of various compliance control techniques extended to cooperative manipulation [7, 1, 8].

The forces or wrenches exerted by cooperatively manipulating robots onto the object transform into internal and external forces, as stated previously, via the *grasp matrix*, which is defined by the geometry of the object. The external force (or wrench) co-vector $\boldsymbol{W}^{\mathbf{o}}_{\text{ext}}$ for a dual-arm cooperative manipulation is given as:

$$\boldsymbol{W}^{\mathbf{o}}_{\text{ext}} = \boldsymbol{A} \begin{bmatrix} (\boldsymbol{W}^{\mathbf{1}}_{\text{ext}})^T \\ (\boldsymbol{W}^{\mathbf{2}}_{\text{ext}})^T \end{bmatrix}^T . \tag{2-20}$$

Here, $\boldsymbol{W}^{\mathbf{1}}_{\text{ext}}$ and $\boldsymbol{W}^{\mathbf{2}}_{\text{ext}}$ are the end-effector wrench of the two arms respectively and $\boldsymbol{A}$ is the grasp matrix. The internal force co-vector $\boldsymbol{W}^{\mathbf{o}}_{\text{int}}$ lies in the null space of the grasp matrix. The overall equation is given as:

$$\begin{bmatrix} (\boldsymbol{W}^{\mathbf{1}}_{\text{ext}})^T \\ (\boldsymbol{W}^{\mathbf{2}}_{\text{ext}})^T \end{bmatrix} = \boldsymbol{A}^{\dagger} (\boldsymbol{W}^{\mathbf{o}}_{\text{ext}})^T + \boldsymbol{B} (\boldsymbol{W}^{\mathbf{o}}_{\text{int}})^T \tag{2-21}$$

where, $\boldsymbol{A}^{\dagger}$ is the pseudo-inverse of the grasp matrix and $\boldsymbol{B}$ spans the null space of $\boldsymbol{A}$ [19, 1]. The internal and external forces acting on any object being cooperatively manipulated could be analysed using Equations (2-20) and (2-21).

The control methods for cooperative manipulation are modified extensions of the compliance control techniques, namely the hybrid position and force control or impedance control. Most of these control strategies are of the centralized form. They are listed as follows:

- **Hybrid position/force based cooperative control:**
  - Symmetric hybrid position/force control Scheme [20, 19]
  - Hybrid external controller [19]
- **Impedance based cooperative control:**
  - Cartesian impedance control via spatial springs [21]
  - External and internal force control via impedance law [1]

The cooperative control method used in this research work is based on ideas from both of the listed impedance based cooperative control strategies. More details on these control methods can be found in the respective citations. A general block diagram [1] describing cooperative manipulation is shown in Figure 2-4, where $\boldsymbol{T}_{\mathbf{o}}$ and $\boldsymbol{T}_{\mathbf{k}}$ are the trajectories of the object and the k^{th} arms respectively. There are two mains points to note from this diagram.

Firstly, cooperative manipulation requires the trajectory of the object being manipulated. This makes sense as arms cannot move independently in such a case. The trajectories of the

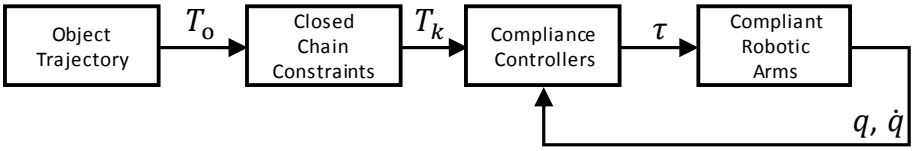


Figure 2-4: A block diagram of cooperative robotic manipulation (adapted from [1]).

arms are dependent on the way the object is required to move. The trajectory of the object should be generated keeping into consideration the kinematic and dynamic limitations of the robotic arms being used.

The second point to note is the closed chain constraints that come from the same fact that the robotic arms are not independent. The relative distance between the arms should always be constant and the velocity zero, to maintain the grasp. These constraints are a consequence of the grasp geometry. Thus, the desired trajectories of the robotic arms are generated from the object trajectory via the closed chain constraints. These constraints are as follows:

$$\boldsymbol{p}_k = \boldsymbol{p}_o + \boldsymbol{R}_o \boldsymbol{p}_k^o \tag{2-22}$$

$$\boldsymbol{R}_k = \boldsymbol{R}_o \tag{2-23}$$

$$\dot{\boldsymbol{p}}_k = \dot{\boldsymbol{p}}_o + \tilde{\boldsymbol{\omega}}_o \boldsymbol{R}_o \boldsymbol{p}_k^o \tag{2-24}$$

$$\boldsymbol{\omega}_k = \boldsymbol{\omega}_o \tag{2-25}$$

where, $\boldsymbol{p}_k$ is the position vector and $\boldsymbol{R}_k$ is the rotation matrix (orientation) of the k^{th} robot, $\boldsymbol{p}_o$ is the position vector and $\boldsymbol{R}_k$ is the rotation matrix of the object, $\boldsymbol{\omega}_k$ and $\boldsymbol{\omega}_o$ are the angular velocities and $\tilde{\boldsymbol{\omega}}$ is the skew-symmetric (cross-product matrix) form of the object angular velocity and $\boldsymbol{p}_k^o$ is the vector joining the k^{th} robot's grasp point and the centre of mass of the robot, also know as the *virtual stick* [22] of the grasp.

Along with these kinematic constraints, some force constraints also need to be met to have the required gripping forces. Pure position control will lead to very high interaction forces and this is where compliance control comes into play [1, 8]. The cooperative manipulation controller used in this research work is based on the previously described spatial springs, but extended to multiple arms, with each arm having a separate controller.

A spatial spring based compliance controller has the end-effector force proportional to the distance between the actual and desired position. This idea can be used to achieve the gripping force needed in a cooperative manipulation grasp by simultaneously satisfying the kinematic constraints required in the manipulation. If length of the virtual stick is reduced by some factor, the desired end-effector position will probably lie somewhere "inside" the object being manipulated, thus constraining its movement, but generating a proportional grasping force. This is shown in Figure 2-5. By varying this desired position and/or the spatial spring stiffness, the correct grasping forces can be achieved without damaging the object [17]. Cooperative manipulation for a dual-arm robot system has been simulated and the results have been described in a later chapter.

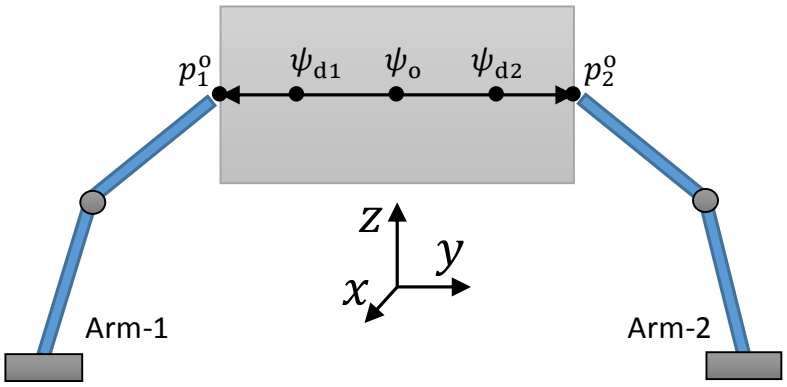


Figure 2-5: A schematic of cooperative robotic manipulation explaining the concept of virtual sticks and the desired end-effector position "inside" the object.

2-7 Sequential Composition

Robots today are becoming more complex by the passing days and can accomplish complex tasks such as folding laundry, cooking, opening a jar and many more. Such complex tasks, which require several different manipulations, which are perfectly timed, can not be achieved using a single controller. These robots consist of a hybrid system framework where various different controllers are used to achieve the goal set of a complex task. One such framework used is *sequential composition control (SCC)* [14, 23, 15].

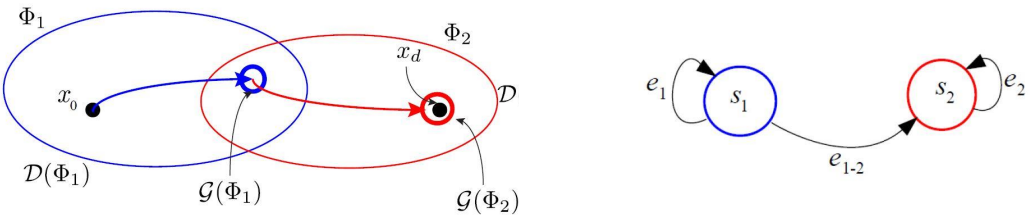


Figure 2-6: (a) Domain of Attraction (DoA) and goal-set of two overlapping controllers with their individual trajectories from an initial condition $\boldsymbol{x}_0$. (b) A hybrid automaton of the two controllers. Images taken from [23].

This framework uses a set of predefined controllers, each of which is active in a particular subset of the state space, known as its *domain of attraction*. The goal-set of each controller lies in the DoA of a "lower" controller, that is, a controllers closer to the final goal-set of the task. Starting from an initial condition that lies in the DoA of one of these overlapping controllers and sequentially switching to a "lower" controller, the required goal state can be achieved successfully which otherwise, is not possible using one single controller. Consider a dynamical system,

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}), \tag{2-26}$$

where $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ is the *state vector* and $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$ is the *input vector* of the system. Defining the control law $\Phi_i : \mathcal{X} \to \mathcal{U}$ with input $\boldsymbol{u} = \Phi_i(x)$, each control law is active only in a subset of the system state space which is the DoA, $\mathcal{D}(\Phi_i) \subseteq \mathcal{X}$ [15, 23]. Every controller also has a *goal set*, $\mathcal{G}(\Phi_i) \in \mathcal{X}$ such that:

$$\forall\ \boldsymbol{x}_0 \in \mathcal{D}(\Phi_i) \Rightarrow \lim_{t\to\infty} \boldsymbol{x}(t) \in \mathcal{G}(\Phi_i), \tag{2-27}$$

where $\boldsymbol{x}_0$ is the initial state vector of the system. This can be visualized as given in Figure 2-6a. This cluster of controllers and their compositions are represented as a finite state machine which is called the *supervisory automaton* [15] and is shown in Figure 2-6b. The controller switching or transition is done based on the relationships among the control policies. SCC is a supervisory controller and its job is to find a sequence of controllers, as defined above, that take the system from any given initial state in the workspace to the final goal-set (obviously assuming that the initial state and the final goal set is in the DoA of one of these controllers).

The idea of SCC can be illustrated using the example of an inverted pendulum [23], as depicted in Figure 2-7a. Consider three different controllers for this pendulum for stabilizing at the upward position (Φ_{up}), downward position (Φ_{down}) and one for swing-up action (Φ_{swing}) respectively. Each controller has its own DoA and a goal set. This has been shown in Figure 2-7b. Now consider a task of bringing the pendulum from a downward position ($\theta_{\text{p}} = \pi$) to the upward position ($\theta_{\text{p}} = 0$).

In order to stabilize the controller at the upward position, the up controller Φ_{up} needs to be activated. But, it can be seen that $\theta_{\text{p}} = \pi$ does not lie in the DoA of Φ_{up}. Hence, to reach the goal set, the swing controllers Φ_{swing} needs to be activated to bring the pendulum states in the DoA of Φ_{up} as the goal set of Φ_{swing} lies in the DoA of Φ_{up}. This action is followed by swinging the controller to Φ_{up} which bring the pendulum to $\theta_{\text{p}} = 0$, which is its goal set. The induced hybrid automaton based on sequential composition is depicted in 2-7c.

The above example of an inverted pendulum only gives a simple insight to sequentially composing various controllers. As the system becomes more complex, the number of controllers also increase. This creates a possibly to achieve tasks which could have been assumed to be impossible to achieve. Unfortunately, this also increases the complexity of deriving the automaton. The process of automatically generating sequential compositions, given a complex task to a system, is a research area which is yet to be studied and exploited.

2-8 Summary

This chapter describes the theoretical and mathematical background of robotic manipulation. Robot kinematic and dynamic modelling and position control were described in brief. Compliance control was introduced and a spatial spring based impedance controller was described in detail. The pros and cons of such a controller were discussed and it was seen that pros outweigh the cons. Cooperative manipulation was explained in some detail along with the geometry and constraints involved in such manipulation. Kinematic and force constraints

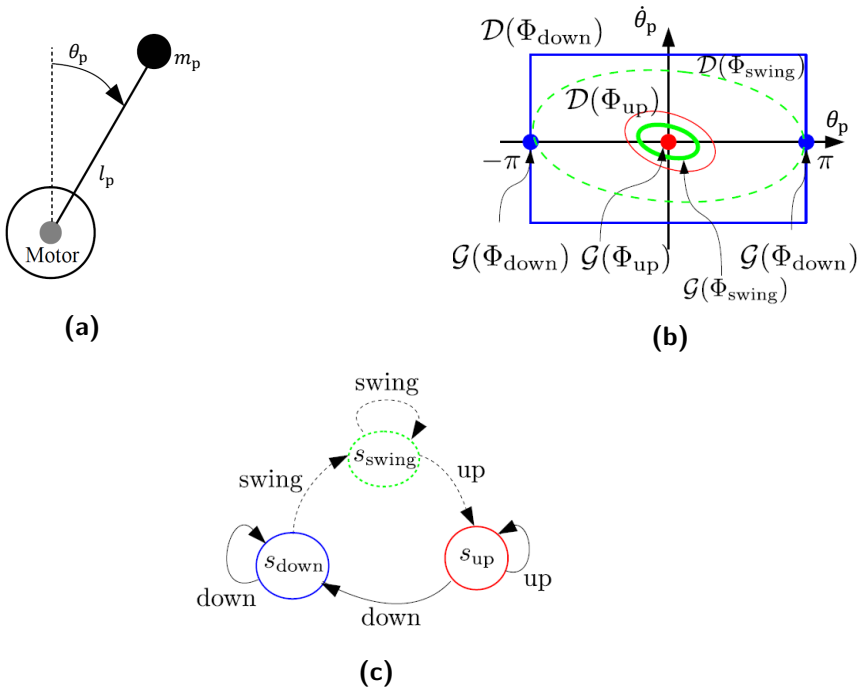


Figure 2-7: (a) Schematic of an inverted pendulum system. (b) DoA and goal-set of the controllers of the inverted pendulum system. (c) A hybrid automaton of the three controllers. Images taken from [23].

that need to be imposed in a cooperative manipulation task were discussed. The concept of sequential composition control was introduced along with a theoretical description. The concepts were explained via a simple example of an inverted pendulum. It was seen that different controllers which are active in separate regions or domains of attractions, could be composed in a way to reach a goal state unachievable by a single controller when the initial state and the goal state lie in the domain of attraction of separate controllers.

Chapter 3

Robot Contact Language

3-1 Introduction

When a robot is commanded to perform a task, it has to carry out various different manipulations and sub-tasks to complete the given task as it is not possible for a robot to achieve every single manipulation using one controller (specification). Tasks generally require manipulating an object, which involves the object being manipulated to have contact with various other components[1] in the manipulation scene like robots, surfaces and other objects. Objects have to be manipulated while maintaining these contacts. Each of these *contact modes* can be represented in the form a language which a robot understands at a symbolic level and subsequently, at a geometric level. A given task can thus, be represented in the form of a symbolic contact language, which will also simplify the task being performed into a simpler control specification, have a structured way of representing a manipulation task and simultaneously bridge the gap between user commands in verbal language to a high level robot language representing a task specification. This is the broad idea behind developing the *Robot Contact Language (RCL)*.

The notion behind RCL be explained with the example of a simple manipulation task of picking and placing an object from one surface to another, as shown in Figure 3-1. This task involves the object initially being in contact with the ground which then makes contact with the two robots for manipulation, lifted up in the air to lose contact with the ground surface, contact being made with the other surface while being picked and placed and finally, losing contact with the two robots after the manipulation is complete. These contact modes can formally be described as a combination of these given scene components using RCL, as will be seen in the following section. Using a few simple rules, relationships to traverse from one contact mode to another can be derived with the help of a *contact map*.

On having generated this contact map, a complex task can be simplified and is just a matter of traversing through this contact map to find the correct manipulation sequence and the contact states it has to go through to achieve the task. Geometrical information related

[1] A component may refer to any object, robot or surface in a given manipulation scene.

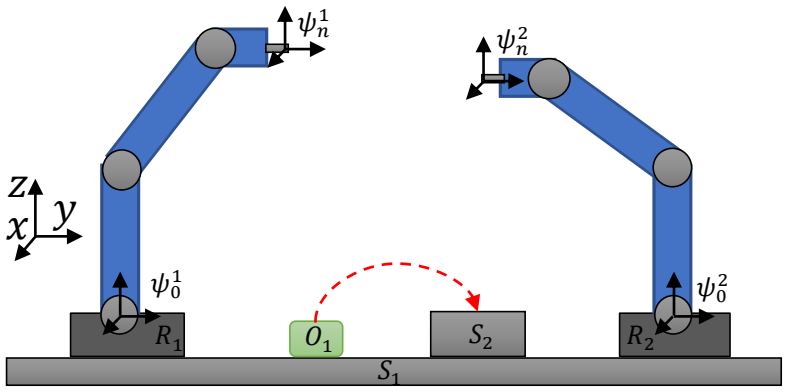


Figure 3-1: Object manipulation - picking and placing object O_1 from a surface S_1 to another surface S_2 using two robots R_1 and R_2.

to components in the scene is added, which updates the spatial relationships amongst the contact modes and further simplifying the map and consequently the search on the map. On having achieved the relevant modes, it is just a matter of carrying out sub-manipulations in these contact modes, bringing the object closer to the final goal and finally, achieving it.

The nomenclature of RCL is described in the first section. The idea of contact maps is then proposed, which is based on some basic rules and the geometrical information available. The following section describes how an object manipulation planning is done using these contact maps. Thereafter, the use of contact maps is described in case of multiple object manipulations.

3-2 Nomenclature of Robot Contact Language

If a person has to command the robots to carry out the manipulation task of picking and placing the object, as shown in Figure 3-1, it would be as following:

" Pick up the green object from the ground and place it on the table. "

From the key-words in this command, one can extract a lot of information regarding the manipulation task. Firstly, the object being manipulated is the green box. The manipulation involves two surfaces, the ground and the table. Initially, the object is in contact with the ground so the robots start to look for it there. The final location of the object is on the table. The information regarding the robotic arms in the scene is certainly available. We thus have all the components present in the given manipulation task.

The components in RCL are classified in 3 broad categories:

- Objects
- Robots
- Surfaces

Objects

These are all movable entities or components present in the scene. Further, there are two types of objects - the one that need to be manipulated and the ones that don't. Every object has a geometry associated with it which is needed to describe its various aspects like how big the object is, where are the stable grasp points, what are the stable placement orientations of the object and where on the object, other objects can be placed (i.e., make contact).

The objects to be manipulated may or may not have a specific position and orientation associated. In the latter case, algorithms can be devised to place the object in a position which satisfies the user command specification. All objects have to be in a stable contact position when not being manipulated (unless specifically commanded).

- An object is represented as $\boldsymbol{O_x}$ in RCL,

where $\boldsymbol{x}$ is the object index varying from 1 to the number of objects in the scene. In this research, the author assumes all objects are similar and have a cubic geometry.

Robots

These are all the robots in the scene that can manipulate objects. These could be any type of robots. The author's research is restricted to robotic manipulators and thus "robots" would indicate robotic manipulators from here onwards. Every robot is associated with a workspace, more accurately, a *dexterous workspace* in which it can position as well orient its end-effector at any arbitrary point (in $SE(3)$).

A robot can also be associated with a pool of different controllers or specific controller with varying parameters (as in the case of this research), which can be set as per the task requirement. It can also be associated with the type of end-effector it is attached to it. Generally, in robotic manipulations, there is a gripper or grasper of some sort, but there are end-effectors for various other purposes. For the purpose of research, the author assumes a flat surface with friction. Hence, one robotic arm can not lift an object. It can only push it around when it is resting on some surface. Hence, cooperative manipulation is required for lifting objects.

- A robot is represented as $\boldsymbol{R_x}$ in RCL.

Surfaces

These represent the surfaces which are static in nature and on which objects can rest in a stable position. These can be floors, tables, etc. Being static, they can not be manipulated or moved by robots. There is no hard boundary between classifying surfaces and objects. A table could also be considered to be an object, rather than a surface. This depends on the capability of the robots and the way they have been to "taught" to identify it as. Similar to objects and robots, surfaces too have geometrical information attached it. The most important being the contact area where other objects can rest in a stable position. Moreover, the volume occupied by a surface can be seen as an obstacle for manipulation planning.

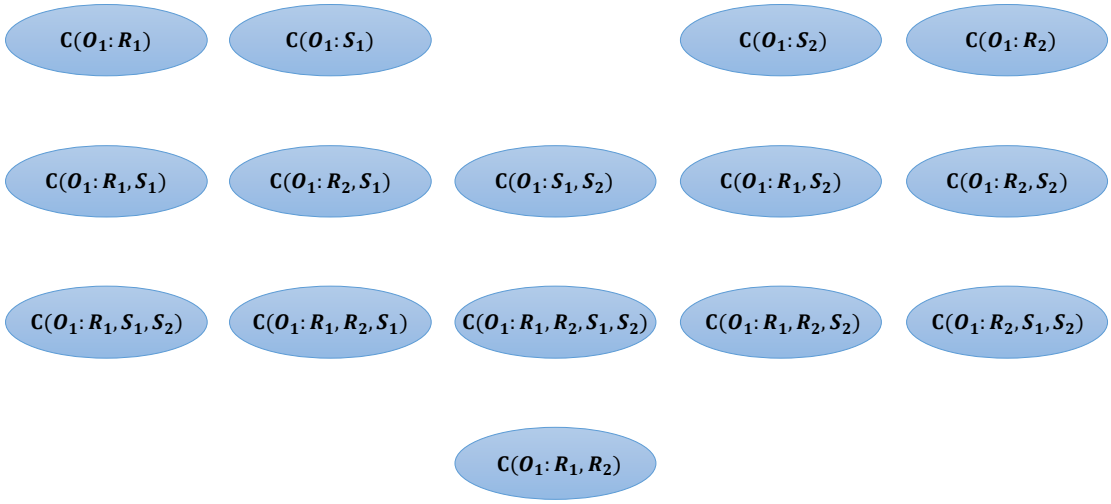


Figure 3-2: $2^{(nt-1)} - 1$ combinations of contact modes for the manipulation task in Figure 3-1, where $nt = 5$.

- A surface is represented as $\boldsymbol{S_x}$ in RCL.

The idea of RCL is to represent a contact mode of objects being manipulated. Let $\boldsymbol{O_m}$ be the object being manipulated by robots in the scene. A contact mode can be written as:

$$\mathbf{C}(\boldsymbol{O_m} : \boldsymbol{R_1}, \boldsymbol{R_2} \ldots \boldsymbol{R_{ns}}, \boldsymbol{O_1}, \boldsymbol{O_2} \ldots \boldsymbol{O_{no-1}}, \boldsymbol{S_1}, \boldsymbol{S_2} \ldots \boldsymbol{S_{ns}}) \tag{3-1}$$

where $\boldsymbol{C}(\boldsymbol{O_m} : \ldots)$ represents the contact of $\boldsymbol{O_m}$ with the components it is followed by. The terms nr, no and ns are the number of robots, objects and surfaces in a given scene. The set of objects $\boldsymbol{O_1}, \boldsymbol{O_2} \ldots \boldsymbol{O_{no-1}}$ obviously does not include $\boldsymbol{O_m}$. It is not necessary that the object is in contact with all scene components. The nomenclature of RCL can understood by the manipulation example of Figure 3-1.

As shown in that figure, the two robots are indexed as $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$, the single green object as $\boldsymbol{O_1}$ and the two surfaces as $\boldsymbol{S_1}$ and $\boldsymbol{S_2}$. For the given manipulation of putting $\boldsymbol{O_1}$ from $\boldsymbol{S_1}$ to $\boldsymbol{S_2}$, the initial and final contact modes can be given as:

$$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1}) \tag{3-2}$$
$$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_2}). \tag{3-3}$$

These two contact modes are static in nature. This means, no manipulation can occur when the object to be manipulated is in this contact mode as there are no robots in contact that can carry that out. These contact modes are generally the initial and final position of an the object. An intermediate manipulation contact mode could be:

$$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2}) \tag{3-4}$$

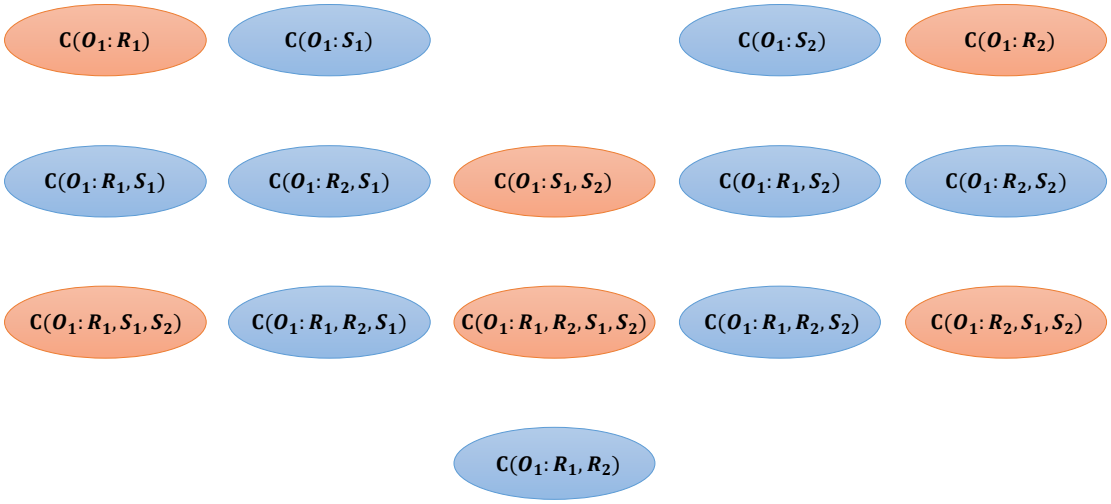


Figure 3-3: $2^{(nt-1)} - 1$ combinations of contact modes for the manipulation task in Figure 3-1, where $nt = 5$. The ones represented red will be discarded due to various assumptions and limitations.

which represents the two robots cooperatively manipulating the object in their common workspace. The total number of contact modes is based on *combinatory logic*. In simple terms, it is just the combination of scene components, excluding the object being manipulated. It can be given as:

$$nc = \sum_{0 \leq k \leq nt-1} \left[\frac{(nt-1)!}{(k!)(nt-1-k)!} \right] - 1 = 2^{(nt-1)} - 1 \tag{3-5}$$

where, nc is the number of combinations and nt is the total number of components in the scene. The negative one in the power is because the object being manipulated needs to be excluded from the formula. The other negative one not in the exponent is so as to not include an empty set of combinations. This formula does seem to suggest that the combinations increase exponentially with the number of objects in the scene. However, many contact modes can be excluded by applying a few assumptions and conditions, which have been cover in the next section. The contact modes for the given example are shown in Figure 3-2 for $nt = 5$, with $\boldsymbol{O_1}$ being manipulated. The relationships amongst the various these contact modes in the scene can be defined by generating a contact map which is also described in the next section.

3-3 Contact Maps

After having generated the various combinations of contact modes, it is yet not possible to carry out a manipulation planning with their help. There have been no relationships defined amongst these various contact modes. Using a few simple rules, transitions can defined amongst these mode and a contact map can be generated. Based on a few assumptions, a few of these combinations can be neglected or removed. Using further information of robot

workspaces, object and surface geometries and the initial and final manipulation positions, many other transitions can be removed due to spatial constraints and other factors.

Taking again, the manipulation example in Figure 3-1, it can be noticed that the two surfaces, $\boldsymbol{S_1}$ and $\boldsymbol{S_2}$ are not connected and it is, therefore, not possible to place $\boldsymbol{O_1}$ on both of these surfaces simultaneously. This condition can be checked via a simple geometric verification. Thus, a mode of the form $\mathbf{C}(\boldsymbol{O_1} : \ldots \boldsymbol{S_1}, \boldsymbol{S_2})$ which has both the surfaces could be neglected. The robotic manipulators used are assumed to have no gripper/grasper at the end-effector. Thus, a single robot can not lift an object on its own and requires cooperative manipulation. Hence, a mode of the form $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_x})$, where x is the robot index, could also be neglected. Similarly, using various other assumptions, more contact modes could be removed and the total set can be reduced to a great extent. Using these aforementioned conditions, a number of contact modes could be omitted for the manipulation task. These are highlighted in red in Figure 3-3 and will be omitted before generating the contact map for that task.

On having the reduced set of contact modes, one can now start defining transitions amongst these contact modes to generate a contact map. This can be done via some simple rules which are as follows:

1. Only one component can make or break contact with the object being manipulated in one transition.
2. Set a maximum limit to number of robots the object can be in contact with.
3. Set a maximum limit to the number of objects it can be in contact with.

Using these rules, a contact map can be generated, with transitions, as shown in Figure 3-4 for the same manipulation example. On inspecting this contact map, it can be seen that there is already a structure in place which can help us analyse a robotic manipulation which include these components.

For the previously stated task, the initial contact mode is $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$ and the goal mode is $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_2})$. As seen from the contact map, there are various possibilities or paths to get $\boldsymbol{O_1}$ from $\boldsymbol{S_1}$ to $\boldsymbol{S_2}$. There is no geometrical information yet attached to this map and thus, the shortest path on this map cannot be determined just yet.

Geometrical information of all scene components are needed in order to practically carry out the manipulation. Starting with robots, their *dexterous workspaces* are required to know the volume of space they can manipulate objects in. Overlapping workspaces of two or more robots will be the workspace where those robots can cooperatively manipulate and also enabling longer connectivity and being able to manipulate objects to longer distances through several robots. Volumes swept by the objects and surfaces serve as obstacles to manipulation path-planning and their exteriors help determine their contact points and hence this information is also very critical.

This information can be extracted using various robot vision techniques using 3D point clouds followed by spatial reasoning to determine and extract various geometrical features of different objects [24]. Probabilistic approaches for semantic mapping have also been studied and applied in [25].

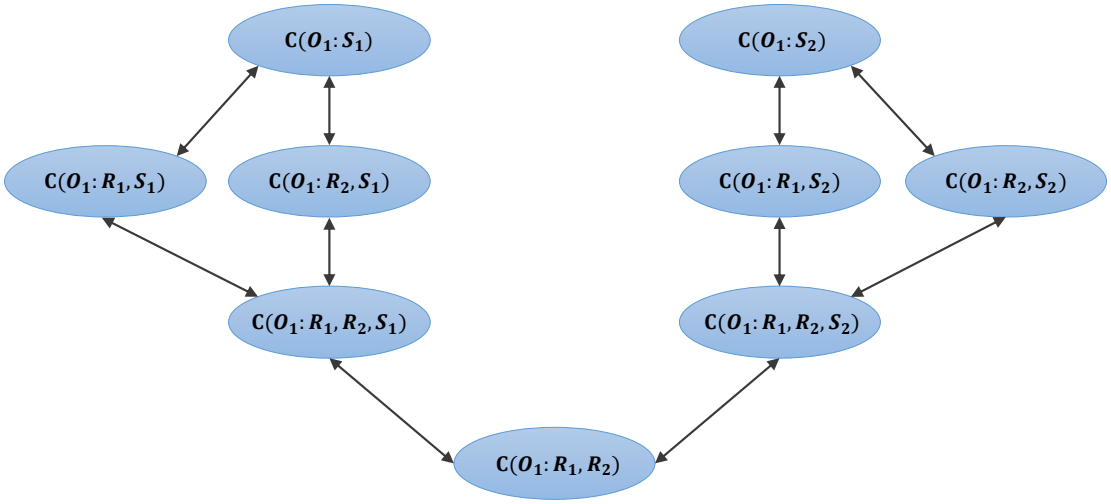


Figure 3-4: Contact map for the manipulation task, as described in Figure 3-1, without spatial relationships.

In the author's research, geometrical information of all the scene components was represented in terms of symbolic inequalities. Using equations of general 3D and 2D volumes and surfaces respectively, these can be represented to a high (enough) degree of accuracy. For example, the workspace of a robot is the volume swept by its end-effector. in the research work, planar robots with all revolute joints were modelled for simulations. The (dexterous) workspace could be written using an equation of a circle. If it needs to be represented using more than one equation, logical **AND**, **OR** and **NOT** operators could be used to represent a set. The equations can be in the form of equalities or inequalities. Similarly, surfaces and objects could also be written using planar equations for their contact points and volumetric equations to identify their volume occupied in space. Figure 3-5 illustrates a planar robot with its workspace in its shaded blue region. The workspace of the robot is given by the equation:

$$y^2 + z^2 < d_0 \textbf{ AND } x = 0 \tag{3-6}$$

with the frame $\mathbf{\Psi_0}$ as the origin. Similarly, for the ground (surface), the contact surface equation is given as $z = 0$ and the obstacle volume to be $z < 0$. Thus, the blue robot workspace, after removing the obstacle workspace, is given as:

$$y^2 + z^2 < d_0 \textbf{ AND } x = 0 \textbf{ AND } z > 0 \tag{3-7}$$

This is obviously a very simple case. The equations could be much more complex with many more logical relationships which could be simplified. This way, geometrical entities are expressed as symbolic expressions and could be evaluated to determine if a point or a path lies within these spaces or not.

The robot workspaces can be given as $\mathcal{W}_{\mathbf{R}}^{\mathbf{1}}$ and $\mathcal{W}_{\mathbf{R}}^{\mathbf{2}}$ for the two robots in the contact map of Figure 3-4. Apart from these, we also need the workspaces of object and surface contact

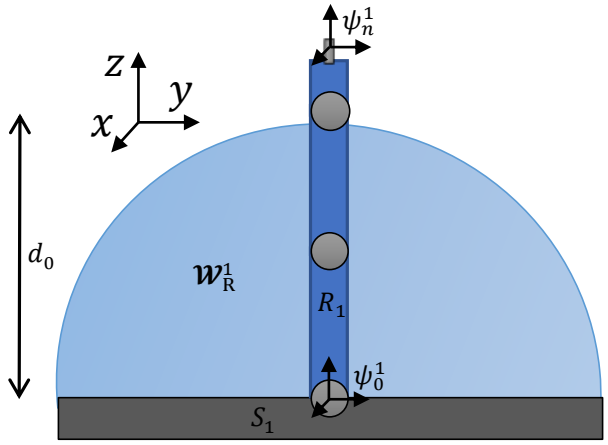


Figure 3-5: An illustration of a robot and its workspace given by the shaded blue region $\mathcal{W}_{\mathbf{R}}^{\mathbf{1}}$.

points. These are given as $\mathcal{W}_{\mathbf{S}}^{\mathbf{1}}$ and $\mathcal{W}_{\mathbf{S}}^{\mathbf{2}}$. Let the total obstacle volume be given $\mathcal{W}_{\mathbf{obs}}$. Using this information, we can assign specific workspaces to each of the contact modes. This has been shown in Table 3-1. The information in the given table tells us the manipulability workspace of that particular contact mode.

Table 3-1: The workspace of each contact mode of the task in Figure 3-1.

Contact mode	Workspace	Manipulable
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$	$\mathcal{W}_{\mathbf{S}}{}^{\mathbf{1}} \cap \neg \mathcal{W}_{\mathbf{obs}}$	No
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_2})$	$\mathcal{W}_{\mathbf{S}}{}^{\mathbf{2}} \cap \neg \mathcal{W}_{\mathbf{obs}}$	No
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2})$	$\mathcal{W}_{\mathbf{R}}{}^{\mathbf{1}} \cap \mathcal{W}_{\mathbf{R}}{}^{\mathbf{2}} \cap \neg \mathcal{W}_{\mathbf{obs}}$	Yes
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$	$\mathcal{W}_{\mathbf{R}}{}^{\mathbf{1}} \cap \mathcal{W}_{\mathbf{S}}{}^{\mathbf{1}} \cap \neg \mathcal{W}_{\mathbf{obs}}$	Yes
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_2})$	$\mathcal{W}_{\mathbf{R}}{}^{\mathbf{1}} \cap \mathcal{W}_{\mathbf{S}}{}^{\mathbf{2}} \cap \neg \mathcal{W}_{\mathbf{obs}}$	Yes
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_2}, \boldsymbol{S_1})$	$\mathcal{W}_{\mathbf{R}}{}^{\mathbf{2}} \cap \mathcal{W}_{\mathbf{S}}{}^{\mathbf{1}} \cap \neg \mathcal{W}_{\mathbf{obs}}$	Yes
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_2}, \boldsymbol{S_2})$	$\mathcal{W}_{\mathbf{R}}{}^{\mathbf{2}} \cap \mathcal{W}_{\mathbf{S}}{}^{\mathbf{2}} \cap \neg \mathcal{W}_{\mathbf{obs}}$	Yes
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2}, \boldsymbol{S_1})$	$\mathcal{W}_{\mathbf{R}}{}^{\mathbf{1}} \cap \mathcal{W}_{\mathbf{R}}{}^{\mathbf{2}} \cap \mathcal{W}_{\mathbf{S}}{}^{\mathbf{1}} \cap \neg \mathcal{W}_{\mathbf{obs}}$	Yes
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2}, \boldsymbol{S_2})$	$\mathcal{W}_{\mathbf{R}}{}^{\mathbf{1}} \cap \mathcal{W}_{\mathbf{R}}{}^{\mathbf{2}} \cap \mathcal{W}_{\mathbf{S}}{}^{\mathbf{2}} \cap \neg \mathcal{W}_{\mathbf{obs}}$	Yes

From the overlapping workspaces of these contact modes, spatial relationships can be defined amongst them which consequently state the workspace within which a transition can happen from one contact mode to another. This has been illustrated in Figure 3-6. All manipulation tasks involving these components could be planned using this final contact map. This has been explained in the next section.

3-4 Manipulation Tasks via Contact Maps

Now that spatial relationships have been defined amongst the various contact modes using geometrical information, we can use the contact map to plan out manipulation tasks. Given

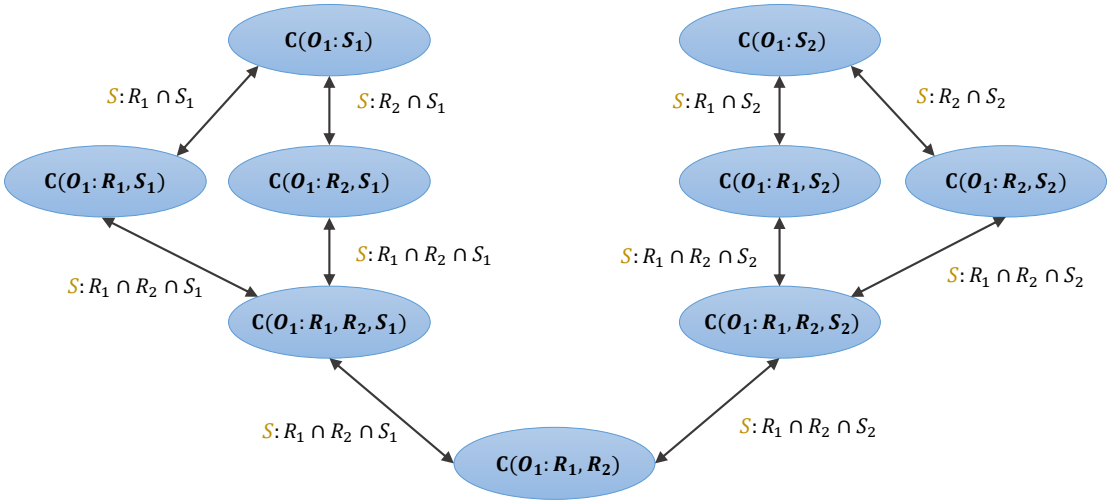


Figure 3-6: Contact map for the manipulation task, as described in Figure 3-1, along with the spatial relationships amongst contact modes.

the initial and final positions of the object to be manipulated, we could traverse through the map to find the shortest path from the contact mode of the initial position to the contact mode of the final or goal position of the object. The previously given example in Figure 3-1 describes a manipulation task of placing $\boldsymbol{O_1}$ from $\boldsymbol{S_1}$ to $\boldsymbol{S_2}$. The initial contact mode is $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$ and the final is $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_2})$.

There are many paths on the contact map from initial to final mode but all are not necessarily valid. A contact mode without any robot is a static or non-manipulable mode, as stated previously. This means that even though all points in its workspace are considered in its workspace, no manipulation can happen when the object is in a static mode. There can be no *intra-mode manipulation*. It is also possible that both, the initial and final object positions lie in the same contact mode. But if that mode is static, there is a need to switch to a manipulable mode to carry out the manipulation and reach the goal point. This happens, for example when the initial and final contact positions are on the same surface. In such a case, the initial and final contact modes are separated by adding a new node in the contact map.

Once this is done, the shortest path from the initial to goal contact mode can be found using one of the most commonly used graph search algorithms for shortest path. The one used in the author's research is the Dijkstra's shortest path algorithm [26]. Many other algorithms like A*, breadth first, depth first, etc. could be applicable too [27]. The shortest path on the contact map tells us what modes the objects needs to go through to achieve the manipulation. Thereafter, manipulation could be planned for each contact mode in this shortest path, having a goal point that lies in the next contact mode and brings the object closer to the global or final goal. The number of manipulations is less than or equal to the number of manipulable modes the object has to pass through in the shortest path. The Dijkstra's shortest path algorithm is as follows.

The contact modes on the contact map are the nodes of the graph. The transitions to other contact modes are considered to be the edges of the map or the graph. The edges could be weighted, but in this application, all weights are equal. The node from which the search starts

is called the initial node which is the initial contact mode and the goal node is the contact mode of the goal position of the object.

There are three variables attached to each node: the distance from initial node, the previous node it came from, and whether it has been visited or not. We could name these variables as **distance**, **from**, and **visited**. The algorithm goes as follows:

1. Initialize the **distance** of all nodes to infinity except the initial node, which is assigned a zero. Set all **weights** to unity.
2. Mark all nodes as unvisited (i.e., **visited** = 0) and set initial node as current node.
3. Calculate the tentative distance to all the neighbours of the current node. (tentative **distance**= edge **weight** + current node **distance**). If the previous **distance** at the neighbour node is greater than the tentative **distance**, replace it with the tentative one.
4. When all neighbours of the current node have been checked, mark the current node as **visited** (i.e., **visited** = 1). This node will never be checked again.
5. End if goal node has been visited (**visited** = 1) or the smallest tentative **distance** amongst the unvisited set of nodes is infinity (i.e., **visited** = 0, in which case, there is no connection from initial to goal node).
6. Else, select the unvisited node with the smallest **distance** and mark it as the current node. Go to step 3.

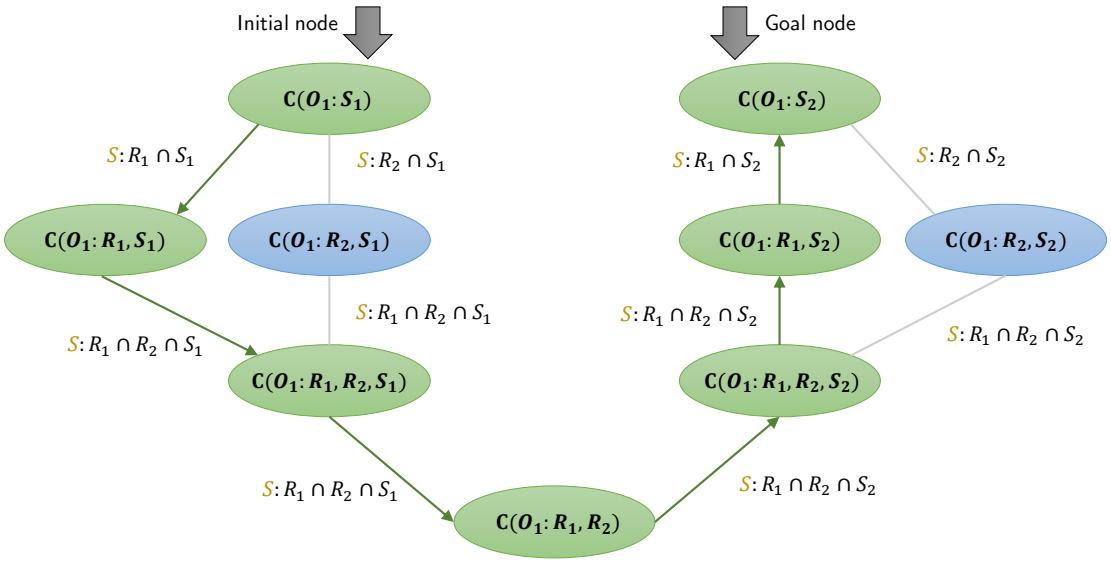


Figure 3-7: Shortest path on the contact map for the manipulation task, as described in Figure 3-1, from the initial to the final contact mode.

The Contact map is passed to an algorithm closely based on Dijkstra's algorithm and it returns the shortest path on the contact map for the given manipulation task. The shortest

path is for the manipulation task in Figure 3-1 is illustrated in Figure 3-7. It is not necessarily exclusive. A different path with the same **distance** is possible. Once the shortest path is available, path-planning is done in each contact mode in that path, followed by controller assignments and sequential execution. This has been described in the next chapter.

3-5 Contact Maps for Multiple Object Manipulation

In the previous section the author cover's the manipulation task of a single object. It was assumed that other objects in the scene, if any, were not supposed to be manipulated. Often that is not the case, especially if it is a humanoid robot working in a domestic environment where it has to place objects on top of each other, organise, de-clutter or make space for objects. It is possible that another object needs to be manipulated first to carry out the original object manipulation or sometimes when there are multiple objects, their sequence of manipulation has to be right. This is because objects have *contact dependencies* with other objects in the scene. There is a need to resolve these object dependencies by correcting estimating the sequence of multiple object manipulation.

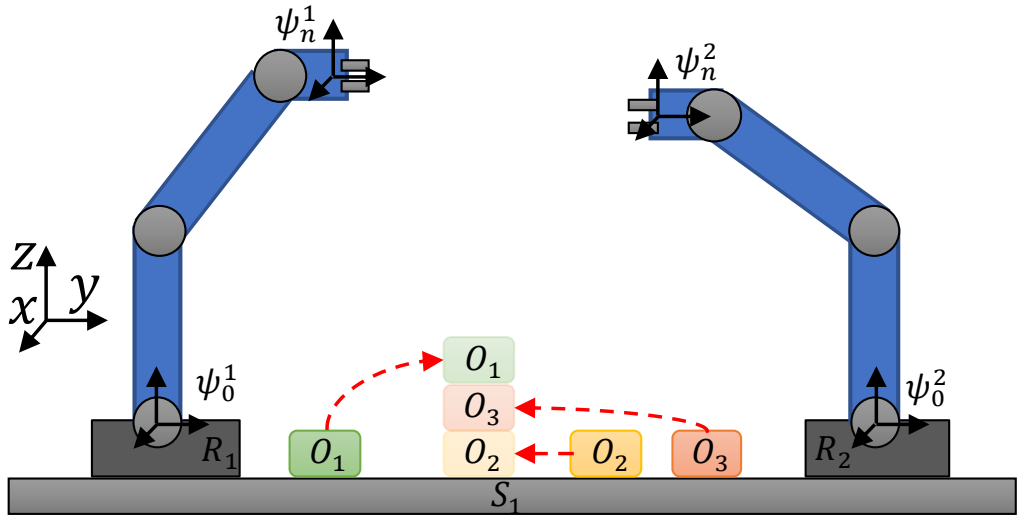


Figure 3-8: A schematic describing a manipulation task of stacking three objects, O_1, O_2 and O_3.

Take for example the multiple object manipulation in Figure 3-8. The three objects need to be in a certain order in their goal position. Hence, the sequence of their manipulation matters. Intuitively, the manipulation sequence would be O_2, followed by O_3 and then O_1. There is no straightforward way for the manipulation planner to decipher this. Hence, an algorithm to correctly find out this sequence is needed. This has been developed in this research work and is know as *Resolve Contact Constraints (RCC)* algorithm. This algorithm uses the author's previously developed concept of contact maps and recursive constraint solving based on the *Resolve Spatial Constraints (RSC)* algorithm developed in [28].

The RSC algorithm of [28] samples future path and recursively uses these samples to construct paths for objects that are blocking. In a manipulation task where an object is blocked by

other objects, these objects have to first be displaced to other locations to free the path for the original object manipulation. The last manipulation is considered to be of the given task. Via path-planning of that manipulation, objects blocking its path could be found. These objects are again analysed in a similar way via path-planning and check the object they are blocked by, and removing them. This way, a recursive algorithm finally gives the paths and the sequence of objects to be manipulated in order to achieve the original manipulation.

The idea of RSC is used to develop the RCC algorithm to find out the correct sequence of object manipulation in a multiple object scenario. This has been explained in the next section.

3-5-1 Obtaining Object Manipulation Sequence

As seen from the manipulation example given in Figure 3-8, there are dependencies amongst objects being manipulated. Here, $\boldsymbol{O_1}$ is dependent on $\boldsymbol{O_3}$ as it rests on it or is in contact with it. Similarly, $\boldsymbol{O_3}$ is dependent on $\boldsymbol{O_2}$. $\boldsymbol{O_2}$ is independent of any other object as it rests solely on $\boldsymbol{S_1}$. If we start our manipulation sequence from $\boldsymbol{O_1}$, we see that in order to manipulate it, $\boldsymbol{O_3}$ should already be at its final position. Moreover, in order to manipulate $\boldsymbol{O_3}$, $\boldsymbol{O_2}$ should be in its final position. As $\boldsymbol{O_2}$ is independent, the manipulation sequence could be $\boldsymbol{O_2}$ followed by $\boldsymbol{O_3}$ and then $\boldsymbol{O_1}$.

The contact maps are generated for one object manipulation while considering other objects to be stationary at their given positions. On finding the shortest path on the contact map, we can acquire the information regarding the dependencies of the object being manipulated with other scene objects. For the object stacking example, if we consider the manipulation of $\boldsymbol{O_1}$, the contact map will have a contact mode of $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{O_3})$ in the map, which states its dependency on $\boldsymbol{O_3}$. Thereafter, it is possible to check if these dependent objects are placed in their correct location or not. It they are not, we have a recursive run of the algorithm to repeat the same procedure. In the end, the object which has no dependency with any objects in the scene will be manipulated first, followed by other objects. Thus, this algorithm resolves the contact constraints and generates the correct sequence of object manipulation without the use of any path-planning algorithm.

This algorithm can also be used along with the RSC algorithm to displace blocking objects for the individual object manipulation. This would produce a very robust algorithm that can carry out various tasks like multiple object manipulations, environment de-cluttering, rescue operations and assembly tasks to name a few. Due to time constraints, RSC could not be implemented along with RCC in this research work. It has been suggested as a part of future work by the author.

The algorithm to acquire the correct object sequence is shown in Algorithm 1 and is called **GetObjectSequence** with RCC as a separate function. In the algorithm, scene geometry information and the initial and goal positions of all the manipulable objects is given as the input. Starting from the first object, the RCC function is called for all the objects. These are indexed with j. This returns the sequence of object manipulation of the j^{th} object along with the objects which it is dependent on, given as $\boldsymbol{LS}$ (which stands for Local Sequence). This is done for every object which is to be manipulated.

When the RCC function is called, it first checks if the object has been placed or not, if it has already been dealt with, it returns an empty sequence. If not, then it first generates

Algorithm 1 GetObjectSequence algorithm with RCC function

Input:
$\mathcal{GE} \leftarrow$ scene geometry
$no \leftarrow$ number of objects
$\boldsymbol{x}^{\mathbf{i}}_{1:no} \leftarrow$ initial object positions
$\boldsymbol{x}^{\mathbf{g}}_{1:no} \leftarrow$ goal object positions

for $j = 1 : no$ **do** ▷ for all objects
  $(\boldsymbol{LS}, \boldsymbol{x}^{\mathbf{i}}_{1:no}, \boldsymbol{x}^{\mathbf{g}}_{1:no}) \leftarrow$ RCC($\mathcal{GE}$, j, $\boldsymbol{x}^{\mathbf{i}}_{1:no}$, $\boldsymbol{x}^{\mathbf{g}}_{1:no}$) ▷ call RCC
  $\boldsymbol{GS} \leftarrow \boldsymbol{GS}$ **append** $\boldsymbol{LS}$ ▷ final manipulation sequence

function RCC($\mathcal{GE}$, j, $\boldsymbol{x}^{\mathbf{i}}_{1:no}$, $\boldsymbol{x}^{\mathbf{g}}_{1:no}$) ▷ RCC function
  if $\boldsymbol{x}^{\mathbf{i}}_{j} = \boldsymbol{x}^{\mathbf{g}}_{j}$ **then**
    return $\emptyset$
  $\boldsymbol{CMap} \leftarrow$ GenerateContactMap($\mathcal{GE}$, j, $\boldsymbol{x}^{\mathbf{i}}_{1:no}$, $\boldsymbol{x}^{\mathbf{g}}_{1:no}$)
  $\boldsymbol{minP} \leftarrow$ FindShortestContactPath($\boldsymbol{CMap}$)
  $\boldsymbol{DO} \leftarrow$ GetDependentObjects($\boldsymbol{minP}$)
  $nd \leftarrow$ length of $\boldsymbol{DO}$
  for $l = 1 : nd$ **do** ▷ for all dependable objects
    if $\boldsymbol{x}^{\mathbf{g}}_{l} \neq \boldsymbol{x}^{\mathbf{i}}_{l}$ **then** ▷ if not in goal position
      $(\boldsymbol{LS}, \boldsymbol{x}^{\mathbf{i}}_{1:no}, \boldsymbol{x}^{\mathbf{g}}_{1:no}) \leftarrow$ RCC($\mathcal{GE}$, $\boldsymbol{DO}^{l}$, $\boldsymbol{x}^{\mathbf{i}}_{1:no}$, $\boldsymbol{x}^{\mathbf{g}}_{1:no}$) ▷ recursion
  $\boldsymbol{LS} \leftarrow \boldsymbol{LS}$ **Append** j
  $\boldsymbol{x}^{\mathbf{g}}_{j} = \boldsymbol{x}^{\mathbf{i}}_{j}$
  return $\boldsymbol{LS}$ ▷ retuning local manipulation sequence

the contact map $\boldsymbol{CMap}$, finds the shortest path $\boldsymbol{minP}$ for the given manipulation and collects the set of all dependable objects $\boldsymbol{DO}$ for that manipulation. It then checks, for the complete set of dependable objects, indexed as l, whether they are correctly placed in their goal position. If not, RCC is recursively called for the unplaced object $\boldsymbol{DO}^{l}$. When there are no dependable objects any more, the object is appended in the manipulation sequence $\boldsymbol{LS}$ and is returned to the previous recursion. The final output of the **GetObjectSequence** algorithm is the sequence of the given multiple object manipulation $\boldsymbol{GS}$, which has been acquired only via contact maps and no path-planning.

3-5-2 Parallel Manipulation

In a scene where there are multiple object manipulations, it is possible that the manipulation specifications would lead to none of the manipulations being dependent on other objects in the scene. It is viable, in such a case, that these manipulations could be carried out simultaneously or in parallel. Take for example the manipulation task in Figure 3-9. $\boldsymbol{O_1}$ and $\boldsymbol{O_2}$ could be manipulated simultaneously. This possibility could also be first seen via the shortest path on their contact maps of the respective objects, as shown in Figure 3-10. The contact mode for a parallel task could thus be written as:

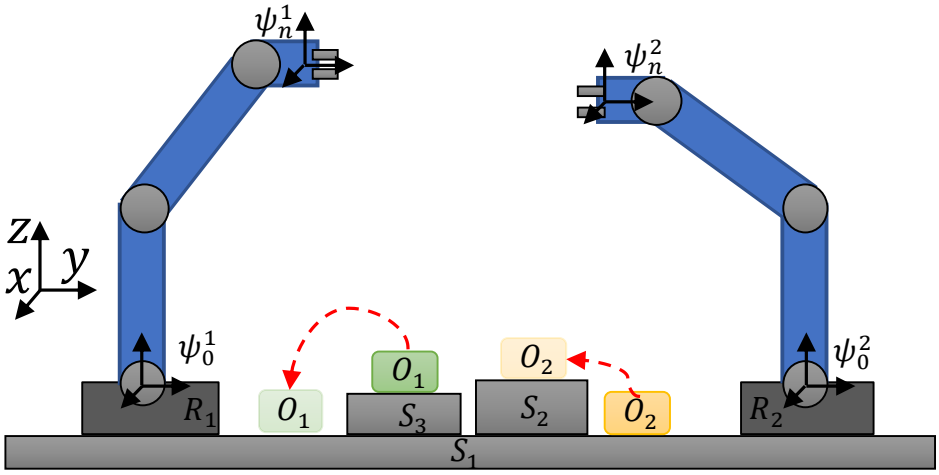


Figure 3-9: Multiple object manipulation tasks that could be carried out in parallel - placing O_1 from S_3 to S_1 and placing O_2 from S_1 to S_2 can be carried out simultaneously via two separate robotic arms in the scene.

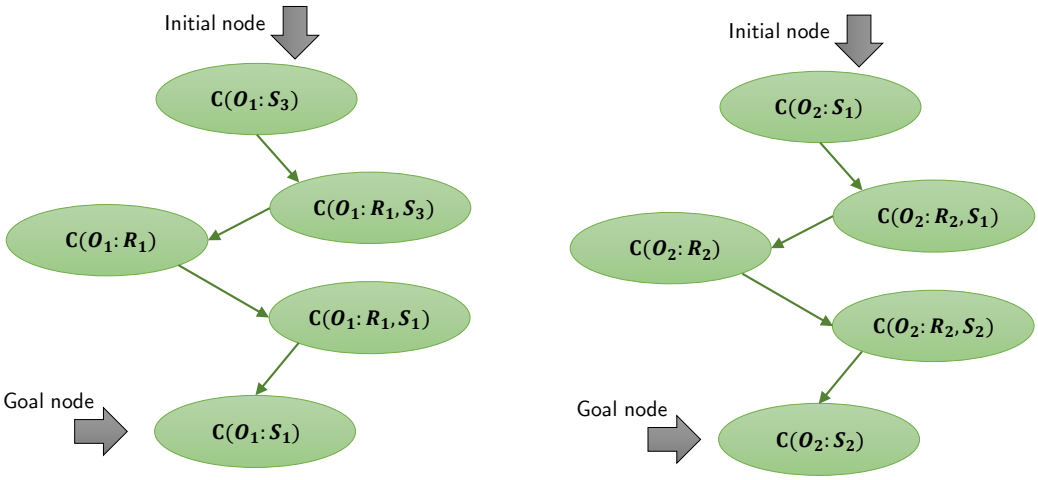


Figure 3-10: Multiple object manipulation tasks that could be carried out in parallel or simultaneously. Shortest path on the contact map for O_1 and O_2 manipulation on the left and right respectively for the task in Figure 3-9.

$$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}) \ \wedge \ \mathbf{C}(\boldsymbol{O_2} : \boldsymbol{R_2}). \tag{3-8}$$

This represents simultaneous manipulation of $\boldsymbol{O_1}$ via $\boldsymbol{R_1}$ and $\boldsymbol{O_2}$ via $\boldsymbol{R_2}$. The parallel mechanism is represented by the logical **AND** operator "$\wedge$". It is possible that these manipulations may experience collision during path planning. The manipulation could then be timed in a fashion that a part of it is running in parallel to save precious work time. The usage of contact maps is to signify a possibility of parallel manipulation, not guarantee it.

3-5-3 Global Contact Map

The idea of contact map described in the previous sections only took the object being manipulated into consideration while building the contact map. It would be convenient and useful to have a complete description of all possible contact modes within one common map. A *global contact map (GCM)* for a manipulation scene can be generated which serves this purpose.

Consider a scene with two objects $\boldsymbol{O_1}$ and $\boldsymbol{O_2}$, two robots $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$, and a ground surface $\boldsymbol{S_1}$. In a global contact map, all possible contact modes are generated from all possible combinations of all the components present, in this case, five. The nomenclature becomes different in this situation as the contact map is not referred to a particular object. Hence, the contact mode of $\boldsymbol{O_1}$ with $\boldsymbol{R_1}$ is given as:

$$\mathbf{C}(\boldsymbol{R_1}, \boldsymbol{O_1}). \tag{3-9}$$

Note that the colon symbol ":" is dropped in GCMs. The total number of combinations for the given five components would then be equal to $nc = 31$. From these thirty-one possibilities, many can be rejected. The combinations of ones does not make sense and are thus dropped. To add to this, all contact modes that do not have an object present, for example, $\mathbf{C}(\boldsymbol{R_1}, \boldsymbol{S_1})$, can also be removed as they do not make sense (apart from special cases). This brings down the total combinations to $nc = 22$. Each of these combinations represents a contact mode in the GCM. Using similar rules of components making or breaking contact, a GCM can created. This has been shown in Figure 3-11.

The interpretation and usage of this map is somewhat different than the previously described local contact maps (LCM). This can be explained using a couple of basic examples. Consider first, a task of manipulating both objects $\boldsymbol{O_1}$ and $\boldsymbol{O_2}$. The task involves stacking of $\boldsymbol{O_1}$ on $\boldsymbol{O_2}$, as shown in Figure 3-12. This is a multiple object manipulation which involves dependencies. Object dependencies can be divided into three broad catagories:

- **No dependency:** There is no contact of the objects being manipulated in any contact mode.
- **Static dependency:** The objects being manipulated are statically in contact with other object(s) in one or more contact modes.
- **Dynamical dependency:** The objects being manipulated are dynamically in contact with each other and need robots to maintain contact.

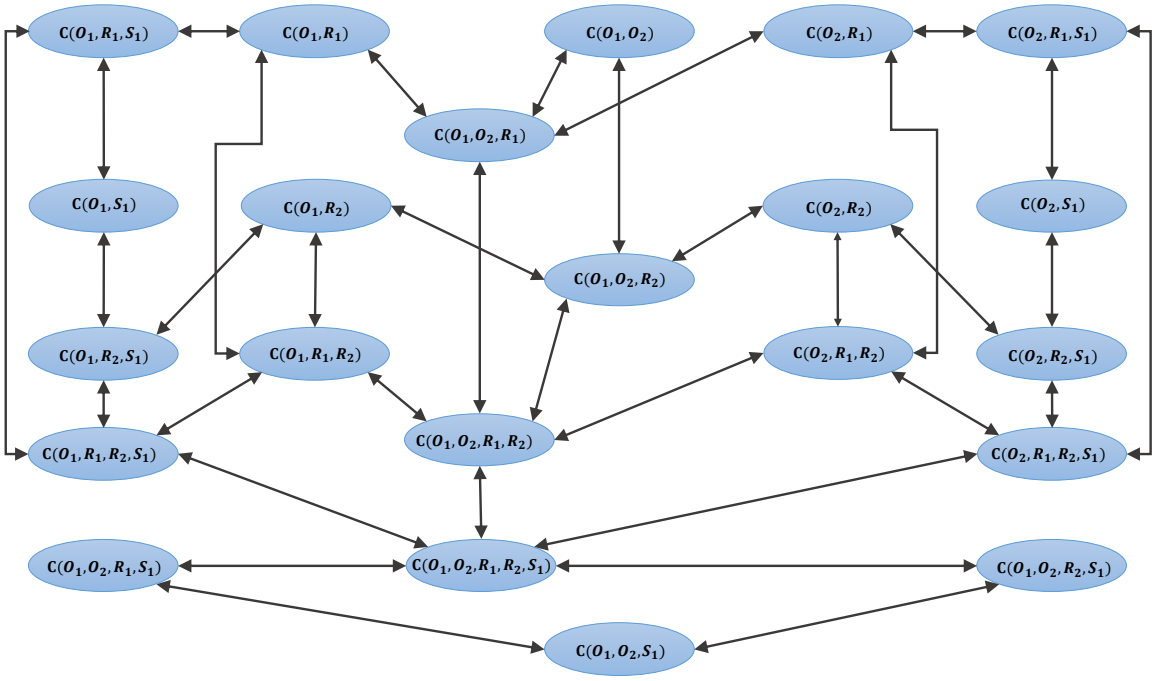


Figure 3-11: A global contact map for five scene components - O_1 O_2, R_1, R_2 and S_1.

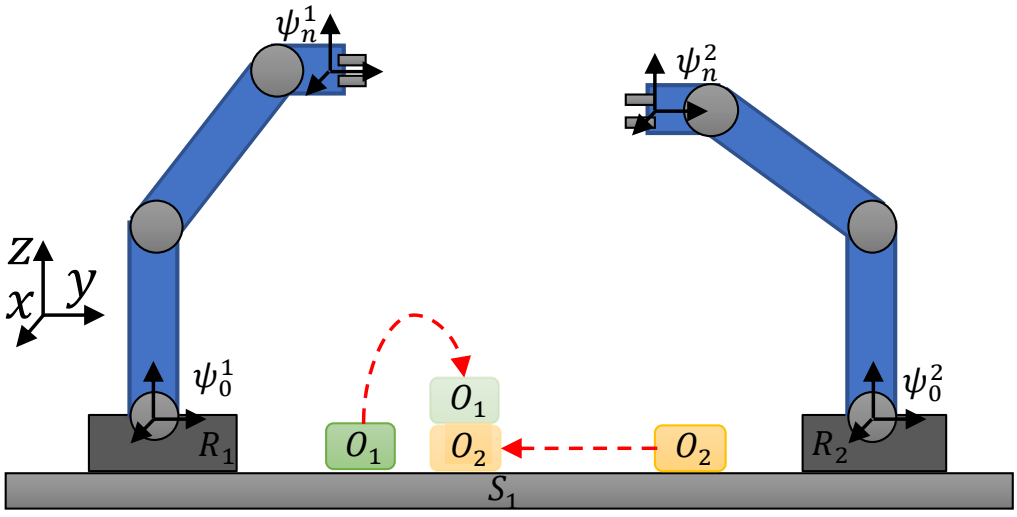


Figure 3-12: A schematic describing a manipulation task of stacking object O_1 on O_2.

In the example given in Figure 3-12, the objects have static dependency. $\boldsymbol{O_2}$ needs to be in correct place first, before $\boldsymbol{O_1}$ can be placed on top of it. Such type of manipulation task can be carried out separately and the sequence of manipulation can be found using Algorithm 1, described in 3-5-1.

The contact modes for the initial and goal positions of $\boldsymbol{O_1}$ are given as $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$ and $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{O_2})$ respectively. The manipulation contact path can be found on the GCM similar to an LCM, as shown in Figure 3-13 with yellow nodes. Similarly, this can be shown for the manipulation of $\boldsymbol{O_2}$ for the same task. The contact modes for $\boldsymbol{O_2}$ manipulation are $\mathbf{C}(\boldsymbol{O_2} : \boldsymbol{S_1})$ for both, initial and goal positions with different contact positions with the surface. The path on GCM is shown by the red nodes.

When there is a manipulation task which involves dynamical dependencies, the usage of GCMs is not similar to LCMs. Such a task is shown in Figure 3-14. Initially the objects are in contact with the ground surface $\boldsymbol{S_1}$. $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ are then used to lift $\boldsymbol{O_1}$ and $\boldsymbol{O_2}$ respectively, and then the two objects make contact.

The local modes for the initial and goal position for $\boldsymbol{O_1}$ manipulation are given as $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$ and $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{O_2})$ respectively. Similarly, for $\boldsymbol{O_2}$, they are $\mathbf{C}(\boldsymbol{O_2} : \boldsymbol{S_1})$ and $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_2}, \boldsymbol{O_1})$ respectively. The global initial mode can be found from the local modes of the two objects. This is done using the **AND** operator and then fusing the components involved, that is,

$$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1}) \wedge \mathbf{C}(\boldsymbol{O_2} : \boldsymbol{S_1}) = \mathbf{C}(\boldsymbol{O_1}, \boldsymbol{O_2}, \boldsymbol{S_1}). \tag{3-10}$$

The right hand side of the equation gives the global contact mode. This can similarly be done for the goal position as well.

$$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{O_2}) \wedge \mathbf{C}(\boldsymbol{O_2} : \boldsymbol{R_2}, \boldsymbol{O_1}) = \mathbf{C}(\boldsymbol{O_1}, \boldsymbol{O_2}, \boldsymbol{R_1}, \boldsymbol{R_2}). \tag{3-11}$$

Hence, the initial and goal global contact modes are $\mathbf{C}(\boldsymbol{O_1}, \boldsymbol{O_2}, \boldsymbol{S_1})$ and $\mathbf{C}(\boldsymbol{O_1}, \boldsymbol{O_2}, \boldsymbol{R_1}, \boldsymbol{R_2})$. A global contact mode can be interpreted as contact amongst the components in any possible combination. $\mathbf{C}(\boldsymbol{O_1}, \boldsymbol{O_2}, \boldsymbol{S_1})$ could be interpreted as:

$$\begin{gathered}\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1}) \wedge \mathbf{C}(\boldsymbol{O_2} : \boldsymbol{S_1}) \\ \mathbf{C}(\boldsymbol{O_1} : \boldsymbol{O_1}) \wedge \mathbf{C}(\boldsymbol{O_2} : \boldsymbol{S_1}) \\ \mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1}) \wedge \mathbf{C}(\boldsymbol{O_2} : \boldsymbol{O_1})\end{gathered}$$

and similarly other possible combinations. The correct local modes are chosen depending on the task being performed. In the global contact map given in Figure 3-11, the path for the task in Figure 3-14 is given by the green nodes.

In each contact mode in the global path, manipulations tasks can be achieved which would bring the objects states closer to the goal states. This can be done using any possible combination of local modes using the components within the global contact mode and is dependent

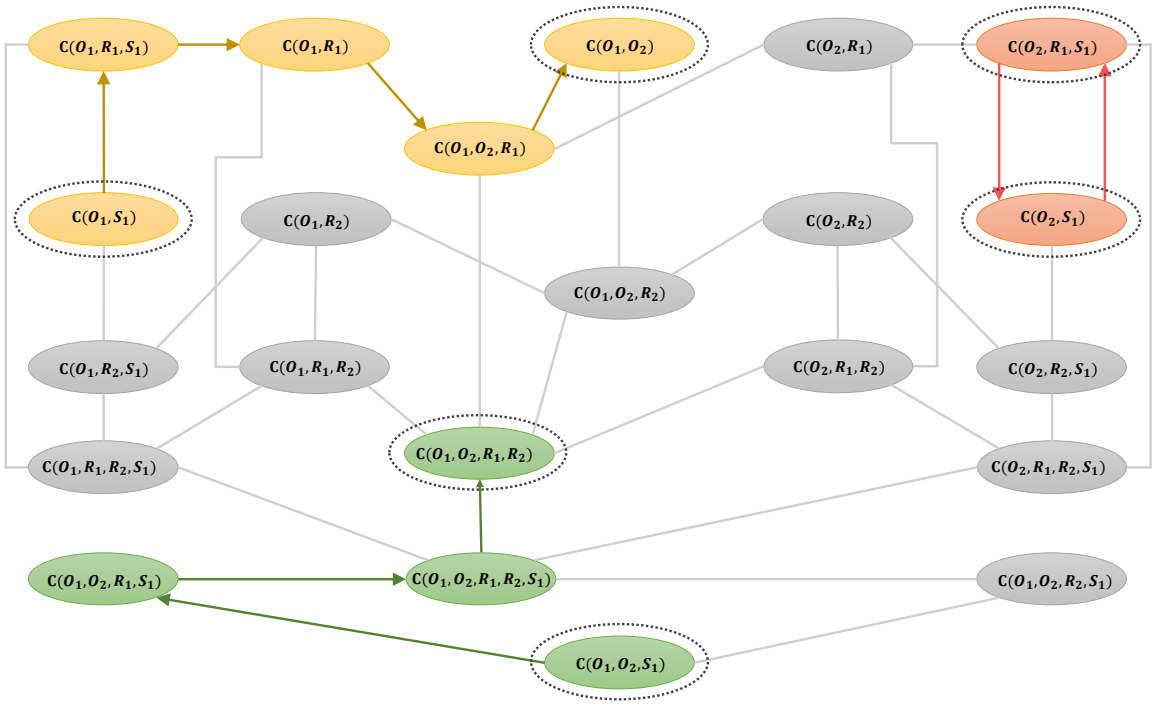


Figure 3-13: A global contact map for five scene components - O_1 O_2, R_1, R_2 and S_1. The yellow nodes show the path for O_1 manipulation and red nodes for O_2 manipulation for the task in Figure 3-12. The green nodes show the global path for O_1 and O_2 manipulation for the task in Figure 3-14.

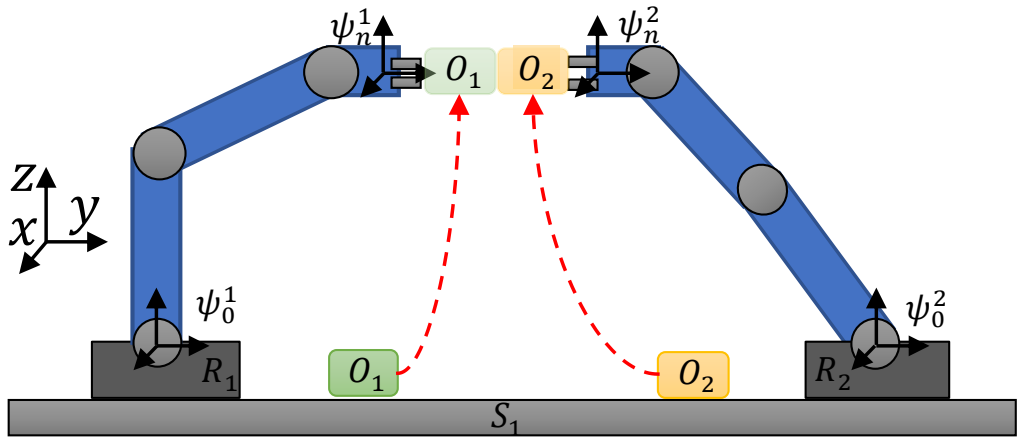


Figure 3-14: A schematic describing a manipulation task of simultaneous manipulation of two objects O_1 and O_2. This causes dynamical dependency between the two objects.

on the given task. Once planning is done in a global contact mode, similar process is undertaken in the next mode of the path. This way, planning is done on a global level for complex manipulations tasks and a complete contact information is embedded in the GCM of a scene for all possible manipulation tasks. The global contact modes and their equivalent local modes for the task in Figure 3-14 are listed in Table 3-2.

Table 3-2: Local and global contact modes of the manipulation task in Figure 3-14.

$\boldsymbol{O_1}$ contact mode	$\boldsymbol{O_2}$ contact mode	global contact mode
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$	$\mathbf{C}(\boldsymbol{O_2} : \boldsymbol{S_1})$	$\mathbf{C}(\boldsymbol{O_1}, \boldsymbol{O_2}, \boldsymbol{S_1})$
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$	$\mathbf{C}(\boldsymbol{O_2} : \boldsymbol{S_1})$	$\mathbf{C}(\boldsymbol{O_1}, \boldsymbol{O_2}, \boldsymbol{R_1}, \boldsymbol{S_1})$
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$	$\mathbf{C}(\boldsymbol{O_2} : \boldsymbol{R_2}, \boldsymbol{S_1})$	$\mathbf{C}(\boldsymbol{O_1}, \boldsymbol{O_2}, \boldsymbol{R_1}, \boldsymbol{R_2}, \boldsymbol{S_1})$
$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1})$	$\mathbf{C}(\boldsymbol{O_2} : \boldsymbol{R_2})$	$\mathbf{C}(\boldsymbol{O_1}, \boldsymbol{O_2}, \boldsymbol{R_1}, \boldsymbol{R_2})$

3-6 Summary

This chapter introduces the concept Robot Contact Language (RCL), its nomenclature and importance in robotic manipulation planning. The idea of contact maps was also introduced which is first built using combinatory logic of all the scene components followed by the addition of geometrical information. This makes it more rich and it is possible to define spatial relationships amongst various contact modes of the map. Using these maps, manipulation tasks could be planned for single object. This idea has been extended to manipulation of multiple object which are dependent on each other. The devised algorithm, Resolve Contact Constraints (RCC), generates the correct object sequence using the contact maps for multiple object manipulation, along with the their individual contact maps and shortest path in their respective contact maps. The idea of global contact maps was also introduced, which embeds complete contact information of all possible manipulation tasks. An example was used to describe its usage for scene objects with both, static and dynamic dependencies.

Chapter 4

Path-planning and Control

4-1 Introduction

After the shortest path on the contact map for a given manipulation has been generated, the next step towards achieving the manipulation task is path-planning followed by controller assignment. Each contact mode in the shortest path represents a sub-manipulation task. Each sub-manipulation task needs path-planning for the object to be manipulated and consequently the robot end-effector. These sub-manipulation paths together form the global path which is used to achieve the overall task.

Path planning could either be done in the workspace or joint space. The complexity in joint spaces increases with the increase in the number of joints. There is also the need of inverse kinematics to map workspace positions and/or targets on to the joint space before planning [4]. As this research work focuses on workspace control, the path-planning too, is in workspace coordinates. This has been described in the next section.

Path-planning is followed by trajectory generation and controller assignment. Trajectory refers to the assigning robot a position, velocity and an acceleration profile with respect to time, that is, the desired end-effector positions and velocities are a function of time. This can be either done online, while execution, or offline by generating a trajectory equation from the given path and kinematic constraints and then the equation is then fed as the desired end-effector positions and velocities with respect to time. More details on path-planning and trajectory generation can be found in [4, 8].

To actually achieve the manipulation tasks, controllers have to be assigned to the robotic manipulators to carry out the tasks. The type of controllers depend on the task being performed. There are several types of manipulation and transition controllers needed for the same. These controllers then need to be sequentially composed and switching condition need to be derived to realize the final control automaton. The process of controller assignment for each manipulation and transition and the consequent sequential composition has also been covered in this chapter.

4-2 Workspace Path-planning

Workspace path-planning refers to manipulation planning in workspace. Before the paths of the robotic manipulators are planned, the path of the object being manipulated needs to be planned. From the contact map shortest path, we have the contact modes and the initial and final position of the object. The objective of the object path-planner is to plan a path separately in each of these contact modes which brings the objects in the workspace $\mathcal{W}$ of the next contact mode and also more closer to the final contact mode. The goal position of the path in each contact mode should:

- lie in the workspace of the next contact mode and
- bring the object closer towards the final goal.

To achieve this objective, a piecewise path-planner should be implemented which satisfy these conditions. The goal position of the path in each contact mode will serve as the initial position of the object for the next contact mode path-planning. This way, the resulting overall path will be a smooth curve.

Ideally, a path-planner is required to plan the path of the object and the robotics arm(s) simultaneously and should take into account the collision of object and the arms with other workspace obstacles along with the self-collision of the arms. In the author's research work, path-planning is done using a modified Dijkstra's shortest path algorithm. It is based on the same algorithm as described in sub-section 3-4 as given in [26]. The path-planner obviously takes into account the collision avoidance of the object with the obstacles in the scene. It is however, only planning the path for the object and not for the robotic manipulators. An obstacle-free path of an object may not guarantee a collision free path of the robotic manipulator. Due to time constraints, path-planning works under the assumption that an obstacle-free path for an object will lead to an obstacle free path for the robot and the simulations are performed in a modelled environment with minimalistic collision probability in order to avoid robot collisions.

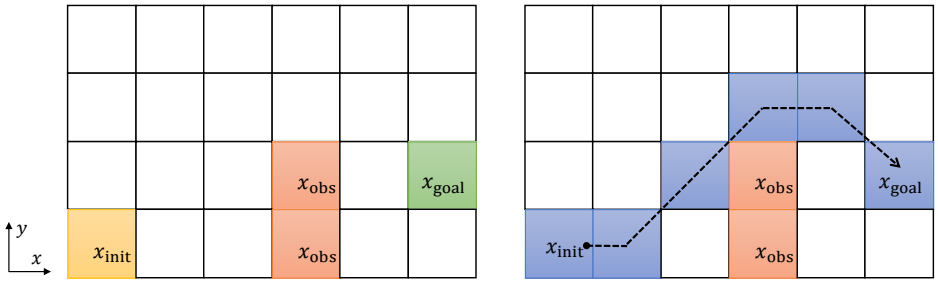


Figure 4-1: Object path-planning using modified Dijkstra's algorithm. The yellow cell is the initial position, green is the goal position and the red ones represent cells occupied by workspace obstacles in the left and right figures. The blue cells in the right figure represent the shortest path.

The basic principle of this algorithm for graph search remains the same. The graph is based on dividing the scene workspace into equal grid samples and performing a search on that

graph. A simplified illustration is shown in Figure 4-1. The grid on the left is the sampled 2D Cartesian space before planing. The yellow block represents the cell of the initial position and the green one represents the goal. The red cells are the ones occupied by the workspace obstacles. The aim is to find the shortest path from the initial position $\boldsymbol{x}_{\text{init}}$ to goal position $\boldsymbol{x}_{\text{goal}}$.

This can easily be done using Dijkstra's algorithm. The distance between two consecutive cells, which is the weight, is kept constant at unity. Every cell, that is not on the border, has 8 neighbouring cells. The distance to every cell is iteratively calculated until the goal cell is reach. The shortest path is traced back once the goal cell is reached.

In a similar way, this method is used to find shortest paths when the overall task has been divided into sub-tasks, which take place in separate, but overlapping workspaces. Take for example the path-planning as shown in Figure 4-2. In this case, it is not possible to find the shortest path from the initial position $\boldsymbol{x}_{\text{init}}$ to final goal position $\boldsymbol{x}_{\text{goal}}$ directly as the workspaces,$\boldsymbol{\mathcal{W}_c^1}$ and $\boldsymbol{\mathcal{W}_c^1}$, are different. This has been demarcated by the orange and green bounding boxes respectively. These two workspaces however, do overlap. If an object needs to be manipulated from $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$, the task can be divided into two sub-tasks. Each sub-task will be to find the shortest path in its respective workspace.

This has been shown in the second (bottom) figure in Figure 4-2. The planning starts from $\boldsymbol{x}_{\text{init}}$ in the first workspace. It obviously cannot reach $\boldsymbol{x}_{\text{goal}}$. This algorithm is however, modified to return the cell position closest to $\boldsymbol{x}_{\text{goal}}$, which is $\boldsymbol{x}^1_{\text{goal}}$. As contact map is also built via overlapping workspaces, $\boldsymbol{x}^1_{\text{goal}}$ automatically lies in the second workspace.

This local goal position $\boldsymbol{x}^1_{\text{goal}}$ will now serve as the initial point for the shortest path search in the next workspace, that is,

$$\boldsymbol{x}^1_{\text{goal}} \sim \boldsymbol{x}^2_{\text{init}}. \tag{4-1}$$

The shortest path can now be found from $\boldsymbol{x}^2_{\text{init}}$ to the cell which is even more closer to $\boldsymbol{x}_{\text{goal}}$. As $\boldsymbol{x}_{\text{goal}}$ belongs to this second workspace, the algorithm will find the shortest path joining $\boldsymbol{x}^2_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$. The complete path from $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$ is shown in the bottom Figure 4-2 with the blue cells. This path is continuous and passes via $\boldsymbol{x}^1_{\text{goal}} \sim \boldsymbol{x}^2_{\text{init}}$.

The same concept is applied to the contact modes in the shortest path of a contact map for a manipulation task. As these contact modes have overlapping workspaces, object path-planning can be done in each of these contact modes to bring the object closer to the final goal, ultimately performing a simultaneous global and local path planning for the given object manipulation. The simulation results for this path planning are shown in the chapter that follows.

This method is also applicable in 3D Cartesian space, but it becomes more computationally expensive as the number of grid cells increase by an exponent. This could be solved by increasing the cell sample size, which would result in faster results as the cost of accuracy. Trade-offs have to made between accuracy and computation time. This algorithm, however does not suffer from getting stuck at a local minimum as it samples and checks the complete space, until the goal cell is reached. If the shortest path exists, it will be found. There are several other path-planning algorithms which work in either joint space or workspace or a combination of both like RRT [29], RRT-connect [30], RRT-JT [31] and potential-field based path-planning [4].

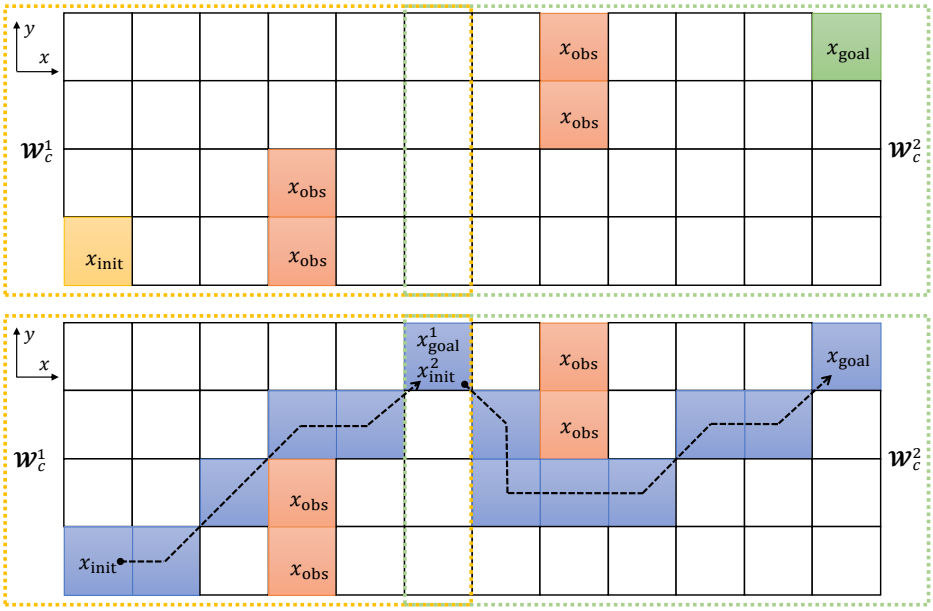


Figure 4-2: Object path-planning using modified Dijkstra's algorithm for sub-tasks with overlapping workspaces. The goal cell of the first workspace is the initial cell for the next workspace path-planning. This cell lies in the overlapping workspace.

4-3 Trajectory Planning

Once the object paths have been generated, a trajectory needs to be assigned to the object for its manipulation. A trajectory, as previously defined, is the evolution of motion with respect to time. In this research work, the trajectories have been kept simple in order to avoid complicated velocity and acceleration profiles. A maximum speed has been assigned to saturate the velocity in case it overshoots. Thus, the velocity is always less than or equal to the maximum value.

The trajectory assigned to objects will determine the trajectory of the robotic arms, in both, single arm manipulation and cooperative manipulation. This is because the arm trajectories are constrained by the object trajectories and will be dependent on them. During position control of the robotic arms, there is no object trajectory attached to it. In this case, the trajectory is generated via the robot initial final positions. Both these cases will be described exclusively. One other thing to note is that the trajectories are generated online from the planned paths while control execution.

When the robotic arms are manipulating an object, their trajectories depend on that of the object being manipulated. The trajectory of the object is generated from its path, planned by the path-planner. The path of an object is given by an array of cell positions in Cartesian space. The desired velocity is kept proportional to the different between the current and goal object position, with an absolute upper limit.

Let $\boldsymbol{x}^{\mathrm{o}}$ and $\boldsymbol{x}_{\mathrm{f}}^{\mathrm{o}}$ be the object current and final position vectors respectively. Let the maximum absolute velocity vector be $\boldsymbol{v}_{\mathrm{max}}^{\mathrm{o}}$. The current velocity is given by:

$$\boldsymbol{v}^{\mathrm{o}} = \min\left(\boldsymbol{v}^{\mathrm{o}}_{\max},\ \boldsymbol{k_1}(|\boldsymbol{x}^{\mathrm{o}} - \boldsymbol{x}^{\mathrm{o}}_{\mathrm{f}}|)\right) \tag{4-2}$$

where $\boldsymbol{k_1}$ is some arbitrary positive constant. On having the desired velocity at every time step from (4-2), the desired object position is calculated via numerical integration using this given velocity and the previous desired position, which is given as:

$$\boldsymbol{x}^{\mathrm{o}}_{\mathrm{des}} = \boldsymbol{x}^{\mathrm{o}}_{\mathrm{des}} + \mathrm{diag}(\boldsymbol{v}^{\mathrm{o}})\ \mathrm{sign}(\boldsymbol{x}^{\mathrm{o}}_{n} - \boldsymbol{x}^{\mathrm{o}})\ dt \tag{4-3}$$

where $\boldsymbol{x}^{\mathrm{o}}_{\mathrm{des}}$ is the desired object position, diag() is the diagonal matrix operator for a vector, sign() is the signum operator, $\boldsymbol{x}^{\mathrm{o}}_{n}$ is the position of the next cell location in the object path vector and dt is the sampling time. On generating the desired object trajectory at every time step, the consequent robot arm trajectory can be generated from the constraint equation given by Equations (2-22) to (2-25).

If the robotic arms are not in manipulation mode, there is no object path involved to generate a trajectory. In this case, the arm trajectories are generated from the current final arm positions $\boldsymbol{x}^{a}$ and $\boldsymbol{x}^{a}_{\mathrm{f}}$ respectively. It is given the same way as in the case of object trajectory similar to equations (4-2) and (4-3) as:

$$\boldsymbol{v}^{a} = \min\left(\boldsymbol{v}^{a}_{\max},\ \boldsymbol{k_1}(|\boldsymbol{x}^{a} - \boldsymbol{x}^{a}_{\mathrm{f}}|)\right) \tag{4-4}$$

$$\boldsymbol{x}^{a}_{\mathrm{des}} = \boldsymbol{x}^{a}_{\mathrm{des}} + \mathrm{diag}(\boldsymbol{v}^{a})\ \mathrm{sign}(\boldsymbol{x}^{a}_{n} - \boldsymbol{x}^{a})\ dt \tag{4-5}$$

with the variables having similar meaning but associated with robotic arm a. This way, robotic arm trajectories are generated from the aforementioned equations online while control execution. This is, again, assuming that there are no obstacles and self-collisions during implementation.

4-4 Controller Assignment

To carry out the manipulation tasks, the robotic arms need specific controllers. The manipulation tasks are broadly divided into two categories - single arm manipulation and cooperative manipulation. Single arm manipulation could be picking and placing an object form one position to another by grasping the object or it could be a robot pulling or pushing the object that is resting on a surface. Cooperative manipulation controllers are a set of controllers functioning individually on two or more arms that are manipulating a common object. These controllers have a common target and their trajectories are constrained by the object trajectory, as seen in the last section.

Another set of controller that are required are transition controllers. Before an object can be manipulated, the robot(s) has to make contact with the object (by grasping or applying a force.) A controller is also needed to make the robot arm orient in a safe workspace after a manipulation task is complete. These transition controllers can be further termed as make contact and break contact controllers. A flow chart classifying the general types of controllers is given in Figure 4-3. This idea was first presented in [32] as *transfer* and *transit* controller. These have been explained a little detail in the following.

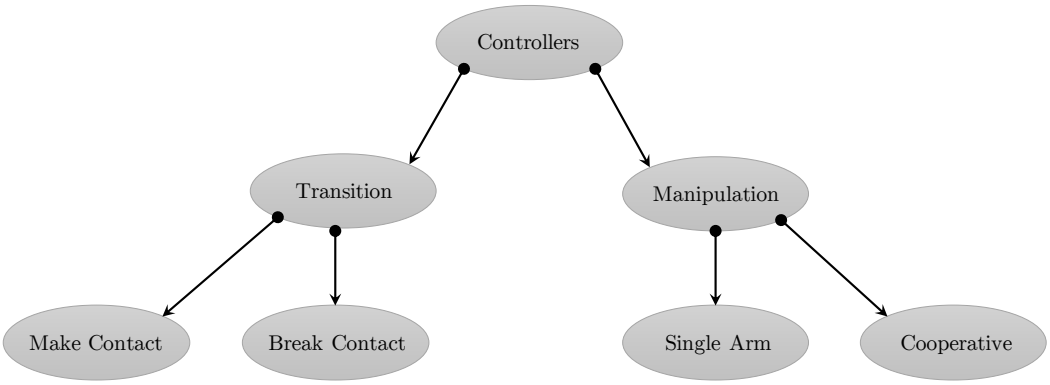


Figure 4-3: Classification of various types of controllers.

4-4-1 Manipulation Controllers

As previously mentioned, manipulation controllers are further divided into single arm and cooperative controllers. It is only possible to switch to these controllers when the robots have already made contact with the object being manipulated, which is in turn, done by the transition controllers.

Single Arm Control

A Single arm controller can either be a robot positioning or placing a grasped object or a force controller which is applying some force or a controller simultaneously applying a constant force while manipulating an object (pushing or pulling). When an object is grasped with an appropriate end-effector, a typical position controller, as explained in sub-section 2-4, could be applied to track a give path or place the object from an initial to goal position. A good property of the spatial springs based controller, as devised in section 2-5, is that it can be used for both, position as well as force control by varying the spatial spring stiffness.

The parameters that need to be defined for the spring stiffness matrix $\boldsymbol{K_d}$ and the Cartesian damping factor $\boldsymbol{D_c}$. It is trivial to infer that a higher value of stiffness will lead to less compliance but a greater position accuracy and the other way around. For position based control, the stiffness is kept high with an appropriate damping factor. Grasping control has not been studied in this research work for simplicity. Hence, single arm manipulation that is studied is that of pushing a box.

Pushing an object at a constant speed requires a constant force on the object being manipulated to compensate for the frictional forces. As stated previously in section 2-5, the end-effector force is proportional to the distance between actual end-effector position and the desired or virtual end-effector position. Hence, the virtual-end effector trajectory is generated in a similar fashion as derived in section 4-3 to apply the a desired velocity profile and the force to push the box. This will no doubt lead to the virtual position of the robot end-effector not coinciding with the object centre of mass or the actual contact point with the object.

Cooperative Control

Cooperative control refers to two or more robots manipulating the same object. Each of these robots have a grasping point where they make contact with the object and apply a certain force. The grasp points should render the object statically stable. The grasp points should not lead to unwanted moments in the objects. The internal and external forces or wrenches on the object, $\boldsymbol{W}^{\mathbf{o}}_{\text{int}}$ and $\boldsymbol{W}^{\mathbf{o}}_{\text{ext}}$ respectively, should implicitly or explicitly be controlled.

In the author's research, robotic arms without any gripper end-effector is used. Thus, the external wrench applied by the robotic arms lead to internal wrench on the object being manipulated. If the robotic arms exactly adhere to the closed chain constraints as given form Equations (2-22) to (2-25), they would only position them at the contact points and will not produce any wrench at the contact point as it is proportional to the distance between the actual and virtual end-effector position of the object.

To counter this problem, as explained previously in section 2-6, the virtual end-effector positions are changed from the actual virtual sticks to modified virtual sticks, which produce a wrench at the contact points as well as maintain the closed chain or grasp constraints. This has also been illustrated in Figure 2-5.

The cooperative manipulation controllers work in two phases after contact is made by all robots. First control sequence is to modify the virtual stick to increase the grasping force, which is done by varying the end-effector virtual positions proportionally for all the robotic arms. On having the desired grasping force, the second control sequence begins which involves actual cooperative manipulation. The robot trajectories are generated from the closed chain constraint equations ((2-22) to (2-25)) while maintaining the grasping forces and the object is cooperatively manipulated.

4-4-2 Transition Controllers

Transition controllers are mainly required in-between manipulation controllers in order to make contact or break contact with the object being manipulated. To demarcate a change of contact mode involving robots, these controllers are used. These have been explained in the following.

Make Contact

A manipulation can not occur unless the robotic arms that are manipulating an object are in contact with it. If the two contact modes for some given arbitrary task are $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$ and $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$ in the order, a manipulation task within the contact mode $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$ can occur only if $\boldsymbol{R_1}$ is in contact with $\boldsymbol{O_1}$. From the previous mode $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$, it is seen that $\boldsymbol{R_1}$ is not in contact with the object. In order to achieve this switching of contact mode, a transition needs to be made where $\boldsymbol{R_1}$ makes contact with $\boldsymbol{O_1}$. This is done using a make-contact transition controller.

This type of controller also involves a two level control sequence. The first is a pure position control where the end-effector places itself close to the object, in line with the point of contact. The orientation of the end-effector is equal to the one during contact. This is followed by a

second control sequence which makes contact with the object to be manipulated. If it is a grasping task, the end-effector moves forward and closes its gripper. If no grasping is involved, the end-effector basically "touches" the object to be manipulated. The consequence of this is that there are grasping forces or an end-effector wrench (measured via a force sensor). This end-effector wrench denotes that contact has been made and that it is possible to switch to the next hybrid state in the control automaton. This concept will be explained later in section 4-5.

The controllers used for this type of control are mainly position based for the first control sequence and a comparatively more compliant controller parameter with a slower velocity profile for the second control sequence.

Break Contact

Similar to making contact, there are transitions in the contact map where a robot leaves or breaks contact with an object that it is manipulating. In such cases, a break contact controller could be used. As an example, take the case opposite to the one in make-contact controller. The task is to go from $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$ to $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$ contact mode. This involves $\boldsymbol{R_1}$ breaking contact with $\boldsymbol{O_1}$, which is achieved by a break contact controller.

This is also achieved by two control sequences. Both involve position control with the first one is of releasing contact or gripping force and moving away in the opposite direction of the grasp point and the second one is to move the end-effector and the robotic arm to a safe position, and freeing up the "active" workspace. When the grasp or contact is released, the grasping force or the end-effector wrench becomes zero.

4-4-3 Control Synthesis

It has been seen that there are various types of controllers that could be used in a given manipulation task. The task of assigning manipulation and transition controllers can be easily done with help of a simple algorithm along with the use of the shortest path on the contact map for the given manipulation task.

For assigning manipulation controllers, each relevant contact mode involved in the manipulation task is checked, starting from the initial one. The number of robots involved in the contact mode are checked at a symbolic level and depending on the number of robots available, an appropriate manipulation controller is assigned. The controller parameter are assigned according to the trajectory of the object and its dynamic and geometric properties.

To assign transition controllers, two consecutive contact modes are checked at a symbolic level for addition or removal of robotic arms; addition represents the need of make contact controller and break contact for subtraction. To make contact, a contact point is required. An object in the scene is always associated with its available contact points or at least its geometry, from which contact or grasp points could be derived. This, unfortunately, is out of the scope of this research work. A general workflow of this process is given in Figure 4-4.

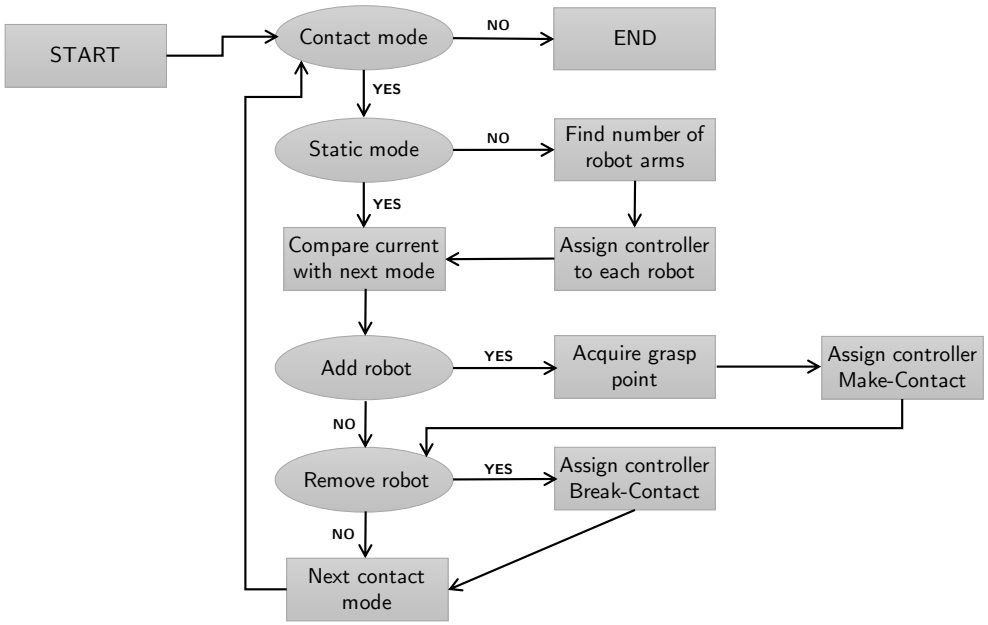


Figure 4-4: The workflow of controller assignment for each manipulation task and the robot transitions that are required.

4-5 Control Execution

A final step to achieving a manipulation task is putting all its controllers together and executing them in a synchronized and hybrid manner. The idea of sequential composition can be applied to bring all these controllers together, compose their domains of attraction and goal-sets and execute the hybrid automaton to achieve the task. It has been seen that the workspaces of the contact modes are already overlapping for a given manipulation task. The path-planning algorithm also works in a way that the goal position of each sub-task or contact mode lies in the workspace of the next one. Hence, the workspaces are already sequentially composed.

The DoA on the other hand, should include all the relevant dynamic information or states in this composition. Along with the workspaces, it should also include the composition of velocities and forces (twists and wrenches) of the robotic manipulators. This means, the DoA should include the twist and wrenches and so should the goal-sets. Intuitively speaking, the velocities of the robotic arms should approach to zero when a sub-task is ending or approaching its goal position (unless it's a special case). As this research aims mainly at manipulation task of picking and placing objects, the velocity or twist part of the goal-set is kept to zero, that is,

$$\boldsymbol{T}_{\boldsymbol{p}}^{\mathbf{g}} = \mathbf{0} \pm \boldsymbol{\epsilon} \tag{4-6}$$

with $\boldsymbol{T}_{\boldsymbol{p}}^{\mathbf{g}} \subset \mathcal{G}(\boldsymbol{\Phi}_p)$. Here, $\boldsymbol{T}_{\boldsymbol{p}}^{\mathbf{g}}$ is goal twist of an arbitrary controller $\boldsymbol{\Phi}_p$. $\mathbf{0}$ is a 6×1 zero vector and $\boldsymbol{\epsilon}$ is very small value (close to machine precision). The twist part of the DoA is

limited by the maximum twist a robotic arm can attain in every direction in $SE(3)$, although this value is not of much use in practical implementation.

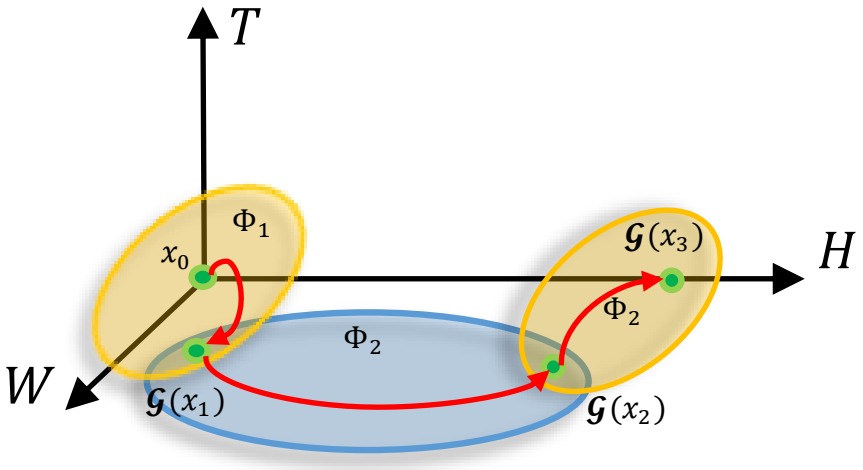


Figure 4-5: A diagram representing sequentially composed controllers. The ellipsoids represent the DoA of the specific controllers and the green points represent the goal-sets. The red-line denotes the trajectory of the robot states.

In the case of the end-effector forces or wrench, the contact of a robot with an object would result in some from of force at the end-effector depending on the way contact is made or the object is grasped. In the author's research work, robot without grippers are studied. Contact is made via end-effectors having a flat surface. This would obviously result in a force acting in the direction normal to the contact surfaces. Assuming the normal surface of the end-effector frame is its z-axis, the wrench part of the goal set, when the object is in contact, would be,

$$\begin{aligned} f_z &< 0 \\ f_z &> -k_1 \end{aligned} \tag{4-7}$$

with $f_z \subset \boldsymbol{W}_p^{\mathbf{g}} \subset \mathcal{G}(\boldsymbol{\Phi}_p)$ where f_z is the force component of the end-effector z-axis and k_1 some arbitrary positive constant. The force sensed in the f_z direction guarantees that the contact with the object is not lost. The force limit set to k_1 would ensure that high forces are not acting on the object being manipulated and if they are, they are detected. Other sensor information like vision could also be used to fuse data and come up with reliable estimates of force values with which contact of robot with an object could be analysed.

This concept of sequential compositions using the Robot Contact Language (RCL) can be further explained using a simple example. Consider a manipulation task with the sequence of contact mode in the contact map shortest path as:

$$\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1}) \rightarrow \mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1}) \rightarrow \mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}) \rightarrow \mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_2}) \rightarrow \mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_2}) \tag{4-8}$$

which represents $\boldsymbol{R_1}$ performing a pick and place task of $\boldsymbol{O_1}$ from $\boldsymbol{S_1}$ to $\boldsymbol{S_2}$. This task involves a lot of controllers. $\boldsymbol{R_1}$ first needs to make contact or grasp $\boldsymbol{O_1}$ and manipulate it and place

it on $\boldsymbol{S_2}$ and finally break contact. The task involves three manipulation controllers for each non-static contact mode and two transit controllers - one for $\boldsymbol{R_1}$ making contact and other for $\boldsymbol{R_1}$ breaking contact in the end.

For representational purposes, we assume that positions, twists and wrenches are all one dimension each. Hence, each of them can be represented in one axis each in 3D space. The sequentially composed controllers for the above mentioned task with their respective DoA and goal-sets are shown in Figure 4-5. Assuming there is no manipulation by the robot on the surfaces, $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$ and $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_2})$ will have no manipulation controllers assigned to them.

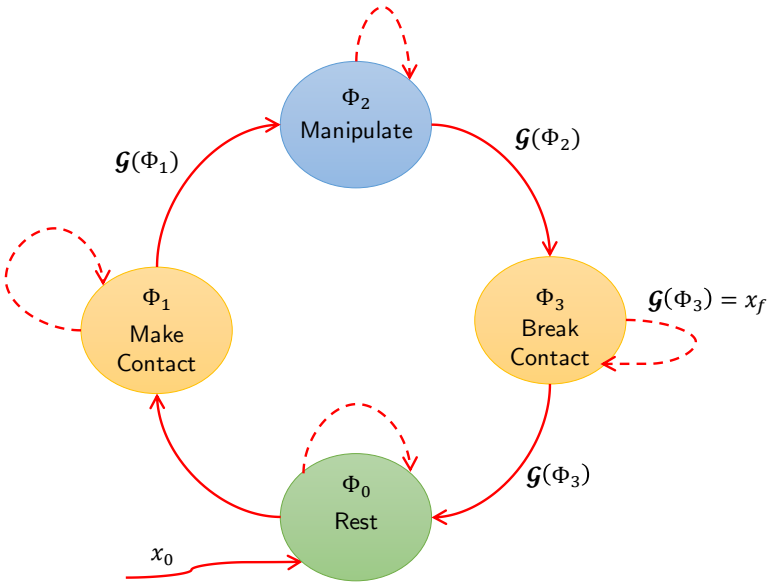


Figure 4-6: The control automaton of the manipulation task described by contact modes in Equation 4-8

.

From the figure, there are a total of three controllers. $\boldsymbol{\Phi_1}$ is a transition controller wherein $\boldsymbol{R_1}$ is making contact. $\boldsymbol{\Phi_2}$ is a manipulation controller which positions $\boldsymbol{O_1}$ from $\boldsymbol{S_1}$ to $\boldsymbol{S_2}$. The last controller $\boldsymbol{\Phi_3}$ is a break contact controller, which breaks contact from the object and places itself is some safe location. The wrench increases when contact is made with the object. Which manipulating, this force is never zero, ensuring no loss of grasping or contact forces. On breaking contact, the force again comes to zero. The manipulation control is mainly responsible for the change in the object's physical position.

In a similar manner, any manipulation task could be executed with the use of contact maps. An involved manipulation task is thus formulated into various sub-tasks and an organised planning using contact maps and smart algorithms assign controllers to each of these sub-tasks which are executed in a hybrid fashion. A hybrid automaton for the previous example is shown in Figure 4-6. Starting from the initial state $\boldsymbol{x_0}$, the three active controllers are sequentially executed. The goal-set of each controller lies in the DoA of the next controller. The goal-set of the last controller is the equal to goal state. On successful execution of these controllers, the overall manipulation task is achieved.

4-6 Summary

In this chapter path-planning of the object being manipulated is discussed. It is seen that with the help of contact maps, the overall path-planning is divided into subtasks. These sub-tasks have their own sub-path for the object manipulation. The sub-paths are derived locally using a modified Dijkstra's algorithm for path-planning in 2D Cartesian workspace.

These sub-paths are assigned specific controllers to achieve the manipulation tasks along with transition controllers to make or break contacts. Switching conditions are assigned for transitions between the controllers. Trajectory planning and generation is done while the sub-tasks are being performed. Whilst sequentially executing these sub-tasks in a hybrid fashion and switching to the controllers that follow, the overall all manipulation task is achieved.

Chapter 5

Simulation Results

5-1 Introduction

In this chapter, various examples of manipulations tasks have been simulated and results have been plotted. Given the initial and goal position of an object to be manipulated, a script generates the contact map, finds the shortest path on it, does workspace path-planning, assigns appropriate controllers and in a hybrid manner, executes the compete manipulation in a simulation software.

The software used to model the robotic arms and perform manipulation is called Virtual Robot Experimentation Platform (VREP) [33]. This has been explained in brief in the following section. Task planning and control are both performed on Matlab. A virtual communication link exists between VREP and Matlab and simulation trigger is done on the Matlab side for each time step after the computations have been performed. Hence, the control can be assumed to be in real-time.

First, the working of a compliance controller is tested using the spatial spring based impedance controller, described in section 2-5. A stiff robot is compared with a fairly compliant robot with both performing the peg-in-the-hole task. This is followed by the results of planning and manipulation of a single object. The last section describes the results of multiple object manipulation tasks and the working of the Resolve Contact Constraints (RCC) algorithm.

5-2 Virtual Robotics Experimentation Platform

Virtual Robotics Experimentation Platform (VREP) is a software specially designed for the modelling and simulation of robotic systems with a dedicated physics engine for dynamical simulations. It offers several robot models in its library and it is also possible to design one's own robots. For the purpose of this research, a three degrees-of-freedom planar robot with revolute joints has been modelled. The robot has no gripper at the end-effector position. It has been equipped with a force sensor at its end-effector to measure forces and torques. For

cooperative manipulation, a similar robot is used with a part of its workspace overlapping with the first robot.

A virtual communication link runs between VREP and Matlab for sending and receiving sensor data and control inputs to and from Matlab respectively. Joint positions and velocities are acquired from VREP for the purpose of control. Other sensor data received are the force sensor readings. The control signal sent to VREP are joint torque values calculated using the control algorithm. The objects being manipulated are all similar and cubic in shape to make their grasping simpler. The simulation screen-shot in Figure 5-1 shows the two robotic arms used for manipulation along with the cubic object.

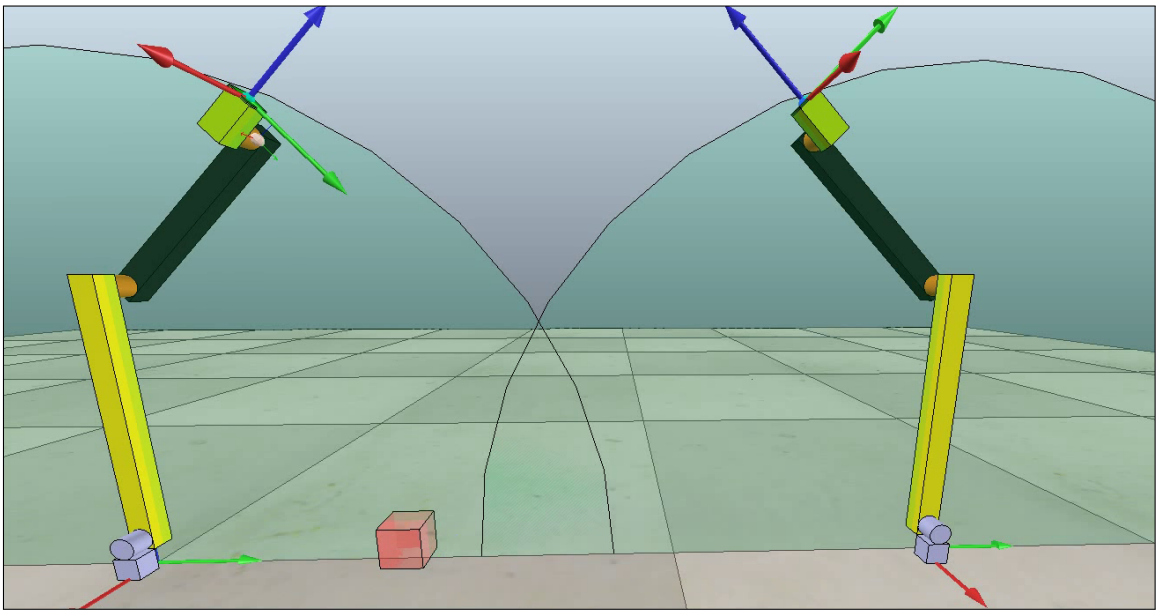


Figure 5-1: Two planar robots and a cubic object in VREP© used for carrying out simulation tests.

5-3 Compliant Manipulation Task

The controller used for both arms is based on the spatial spring controller as described in section 2-5. By varying the stiffness, both position and force control can be achieved. A higher stiffness would lead to higher position accuracy with a less compliant robot end-effector and vice-versa. To test the usability and versatility of this controller, a test was performed using a stiff and a compliant manipulator. The task was similar to the peg-in-the-hole task, described in section 2-5.

Figure 5-2 shows the initial and final states (top and bottom respectively) of the robots during the execution of the task. The robotic arm on the left is a stiff robot with the stiffness parameters set to a considerably high value. In contrast, the robot on the right changes its parameters to become much more compliant when it is close to the insertion. The test conditions for both the arms are given as follows:

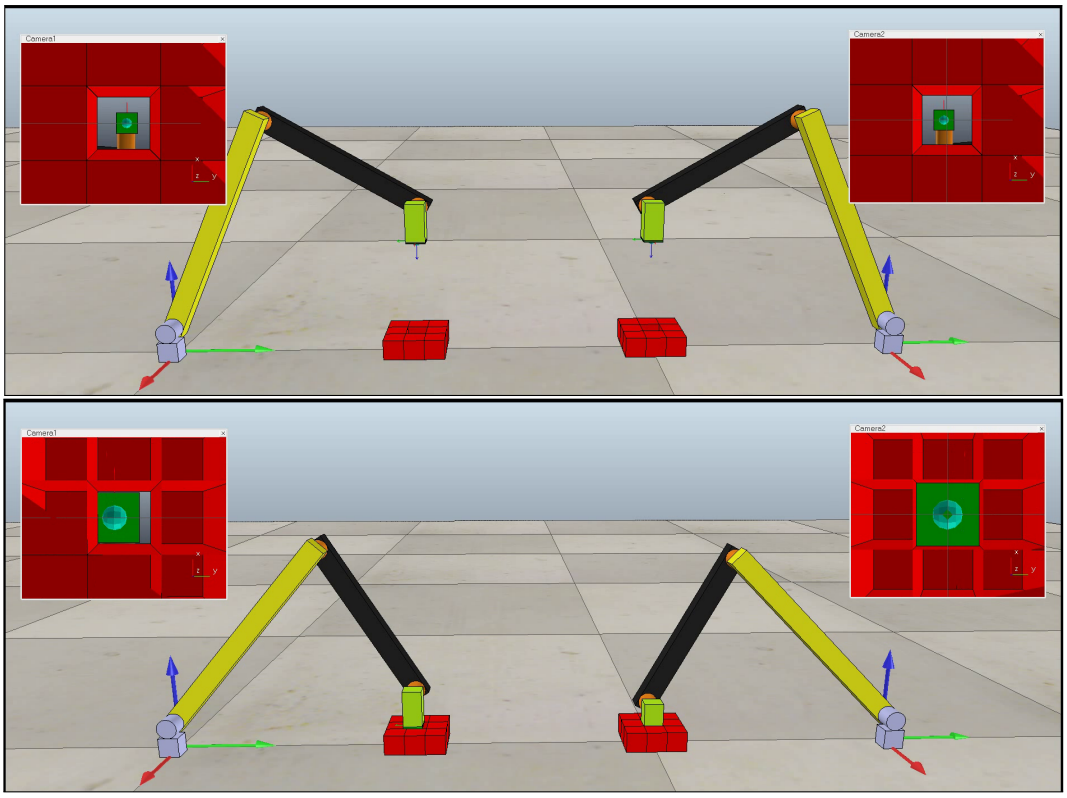


Figure 5-2: Peg-in-the-hole task being executed in VREP.

- The global base frame $\mathbf{\Psi_0}$ is attached to the base of the left robotic arm. All measurements are relative to this inertial frame.
- Both the arms start from rest with their initial end-effector position and orientation (6D) vectors as:

$$\begin{bmatrix}(\boldsymbol{\theta}^{\mathbf{1}}_{\mathbf{init}})^T & (\boldsymbol{p}^{\mathbf{1}}_{\mathbf{init}})^T\end{bmatrix}^T = \begin{bmatrix}-\pi & 0 & 0 & 0 & 0 & 0.25 & 0.12\end{bmatrix}^T$$
$$\begin{bmatrix}(\boldsymbol{\theta}^{\mathbf{2}}_{\mathbf{init}})^T & (\boldsymbol{p}^{\mathbf{2}}_{\mathbf{init}})^T\end{bmatrix}^T = \begin{bmatrix}\pi & 0 & 0 & 0 & 0 & 0.5 & 0.12\end{bmatrix}^T$$

  where one and two are the stiff and compliant arms respectively. Both these positions are with respect to a common base frame. $\boldsymbol{\theta}_{\mathbf{init}}$ is the initial orientation vector given by Euler angles and $\boldsymbol{p}_{\mathbf{init}}$ is the position vector in Cartesian space.
- The holes are positioned on the ground surface. The final or goal positions of the end-effectors to correctly enter the holes are:

$$\begin{bmatrix}(\boldsymbol{\theta}^{\mathbf{1}}_{\mathbf{goal}})^T & (\boldsymbol{p}^{\mathbf{1}}_{\mathbf{goal}})^T\end{bmatrix}^T = \begin{bmatrix}-\pi & 0 & 0 & 0 & 0 & 0.25 & 0\end{bmatrix}^T$$
$$\begin{bmatrix}(\boldsymbol{\theta}^{\mathbf{2}}_{\mathbf{goal}})^T & (\boldsymbol{p}^{\mathbf{2}}_{\mathbf{goal}})^T\end{bmatrix}^T = \begin{bmatrix}\pi & 0 & 0 & 0 & 0 & 0.5 & 0\end{bmatrix}^T.$$

  The trajectory would be a straight line in the downwards position.

- If path-planning and trajectory tracking for this task are perfect, both the robotic arms would achieve the task. To see the effect of position error in the constrained directions, the goal positions was given a small mismatch from the actual position in the y-axis direction. The y goal positions were set to be 0.245 m and 0.505 m instead of 0.25 m and 0.5 m for arms one and two respectively. Thus, the error was 5 mm.
- Both the arms commenced with high stiffness parameters. When the second arm reached a z-axis position of less than 0.02 m, its parameters changed to make the arm highly complaint.

From Figure 5-2, it is seen that the stiff robotic arm is unsuccessful in inserting the end-effector into the hole due to the position error while the compliant robot is able to achieve the task. This is because it is not "stubborn" to accurately position itself. It can wiggle around the goal position and the forces felt due to mismatch in the hole position make the arm move towards the hole and thus achieving the task. Quantitative results are shown and described in the description that follows.

Comparison of Forces

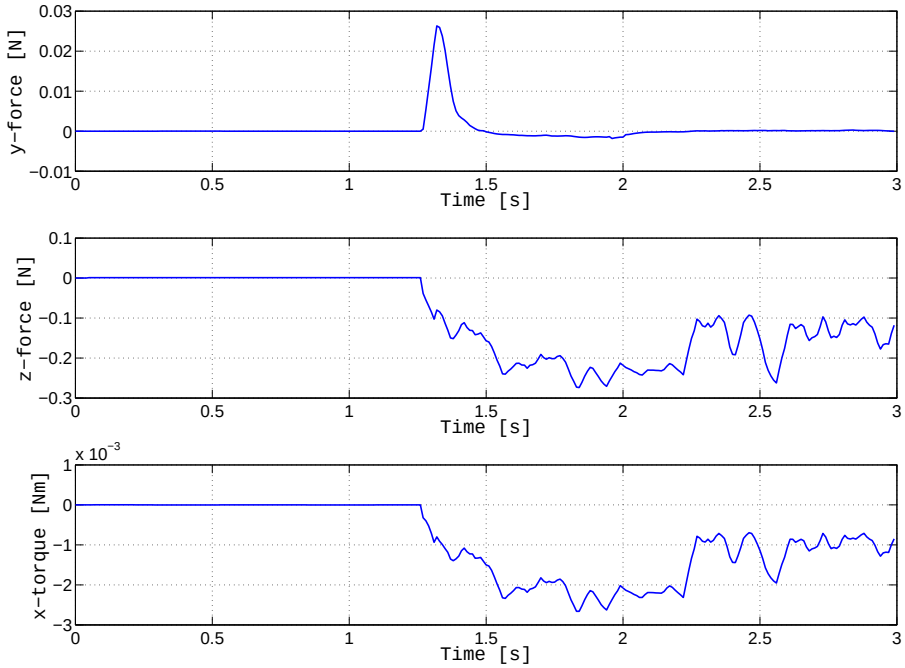


Figure 5-3: Forces acting at the end-effector for peg-in-the-hole task with stiff robotic arm.

The end-effector is attached with a force sensor which can measure the external forces or wrench acting on it. As the robots are planar, the relevant position axes are y and z and orientation axis is x. On having stated that, the forces and torque acting on and about these three axes for both the arms have been measured for the peg-in-the-hole task.

Figure 5-3 gives the forces acting on the stiff arm. More stiffness leads to higher interaction forces. Initially, the forces are zero due no interaction of the end-effector with any external object. At around 1.25 seconds, the arm makes contact with the hole and it is not aligned exactly with the vertical axis of the hole due to the offset in the goal position. The peaks are caused due to the initial interaction and the momentum of the arm as it is approaching with a great speed.

The most significant forces are acting in the z-axis frame of the end-effector. This is the direction normal to the end-effector and will experience the maximum force during impact and otherwise. The force in this direction is in the range of -0.15 N to -0.25 N. As the goal position is not reached, the end-effector remains constrained. The forces are proportional to the actual and virtual end-effector position and thus, the interaction forces do not die out.

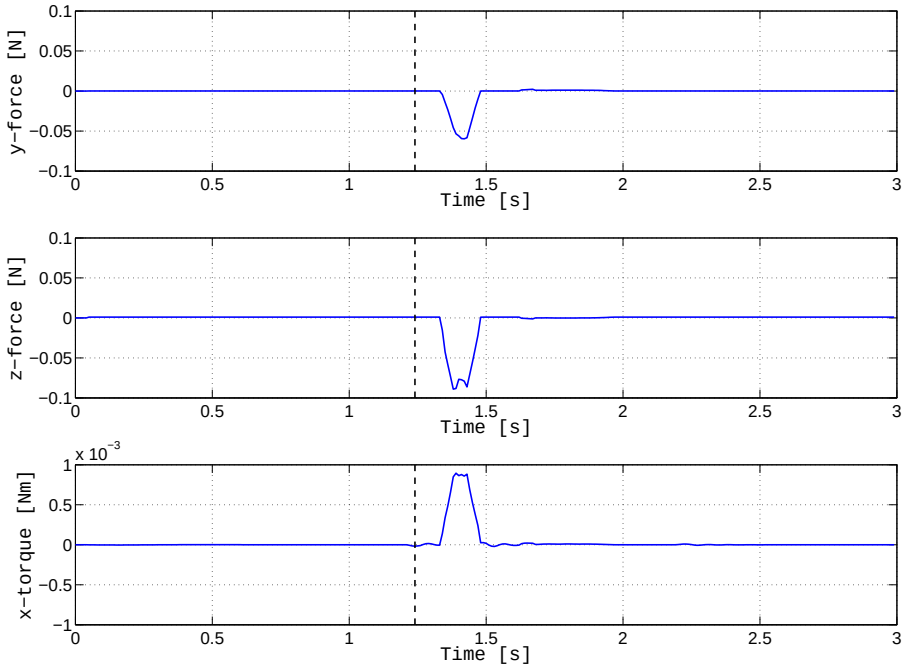


Figure 5-4: Forces acting at the end-effector for peg-in-the-hole task with compliant robotic arm.

In Figure 5-4, the same interaction forces acting on the other arm have been displayed. The dotted vertical line represents the time instance at which the stiffness parameters are changed to make the arm more compliant. This is done when the end-effector is approaching the hole. As in the previous arm, the initial forces are zero as there is no interaction with any external environment.

When the end-effector touches the body around the hole position. Due to position inaccuracy, the insertion will not directly take place. There will be interaction forces acting at the end-effector. As the robot arm is more compliant, the interaction forces are considerably low. The z-axis force is now less than 0.01 N (in an absolute sense) and short lived.

Due to the arm being compliant, position accuracy gets compromised to some extent. This is not exactly a disadvantage and that is what is making the robot move around more freely

and and slip into the hole position without high interaction forces. In this case, the task is achieved and the end-effector reaches the goal position. Thus, after the initial peak, the interaction forces go back to zero again.

Comparison of Position Errors

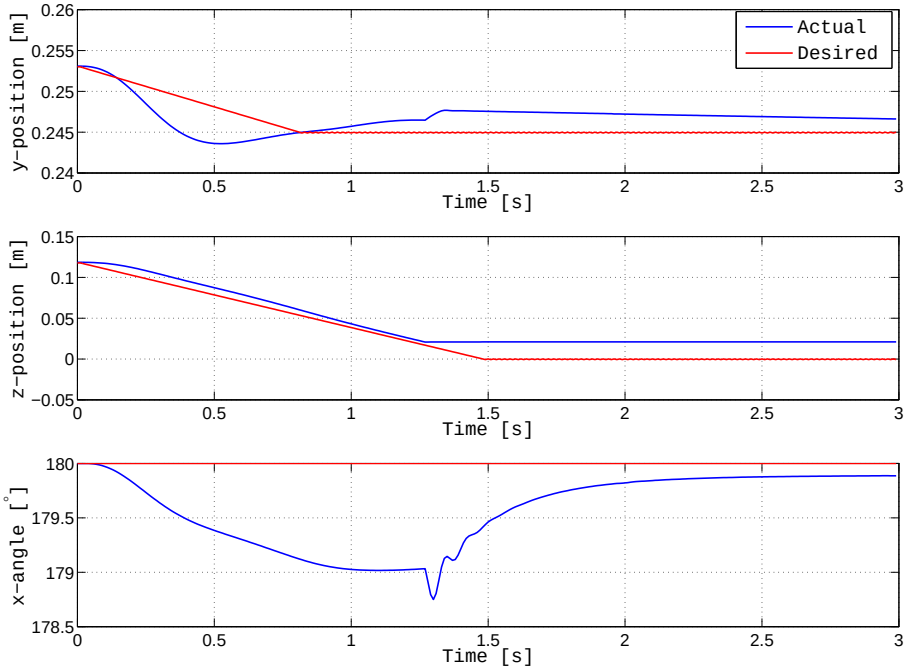


Figure 5-5: Trajectory tracking of the end-effector for peg-in-the-hole task with stiff robotic arm.

A constant velocity trajectory is generated for the desired end-effector position in each direction for both the arms to reach from their initial to goal position. The goal positions, as previously mentioned, have an offset of 5 mm in y direction. The desired orientation remains constant at a value of 180 deg and -180 deg for both arms respectively.

Figure 5-5 shows the position profiles of the stiff arm. Due to sudden acceleration at the beginning, there is some tracking error which slowly approaches zero as the end-effector starts tracking the desired position. The red line is the desired end-effector position and the blue one is the actual. After the impact, the stiff arms is unable to reach its goal position inside the hole. It gets stuck on the outer surface of the hole and remains there. This can be seen by the constant or steady-state position error after 1.5 seconds.

Figure 5-6 shows the position errors with time. The error along y axis are considerably small. The maximum position errors are along the z-axis, as expected. The steady-state error along this axis is 0.02 m which is the height of the hole structure. The orientation error about x-axis at steady state is less that 0.2 deg and a maximum tracking error of less than 1.5 deg.

In the case of complaint arm, the steady position errors are much less as the goal position is reached. Figure 5-7 shows the actual and desired end-effector position for the compliant arm,

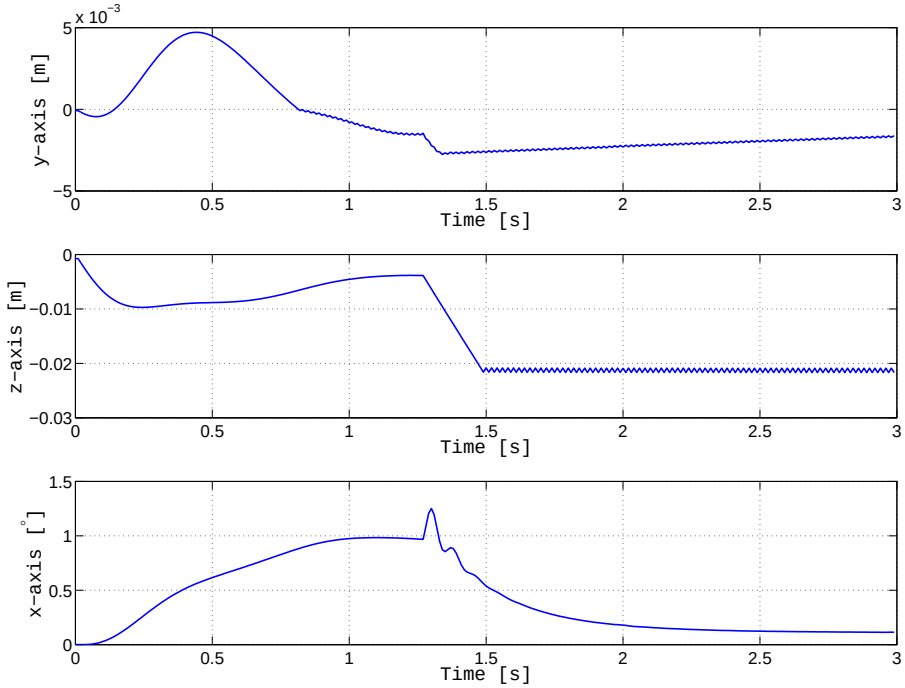


Figure 5-6: Position error of the end-effector for peg-in-the-hole task with stiff robotic arm.

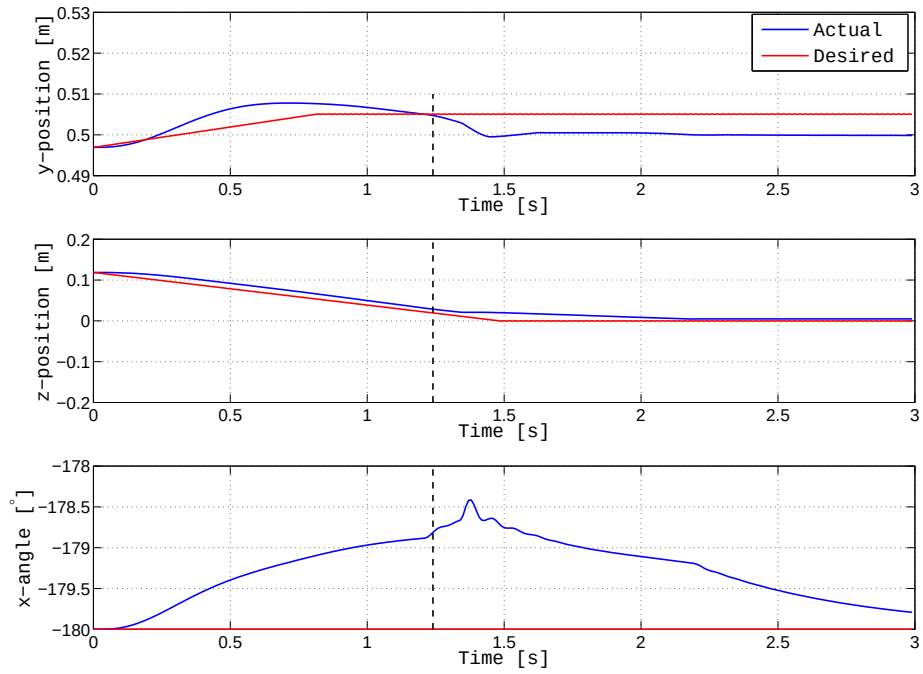


Figure 5-7: Trajectory tracking of the end-effector for peg-in-the-hole task with compliant robotic arm.

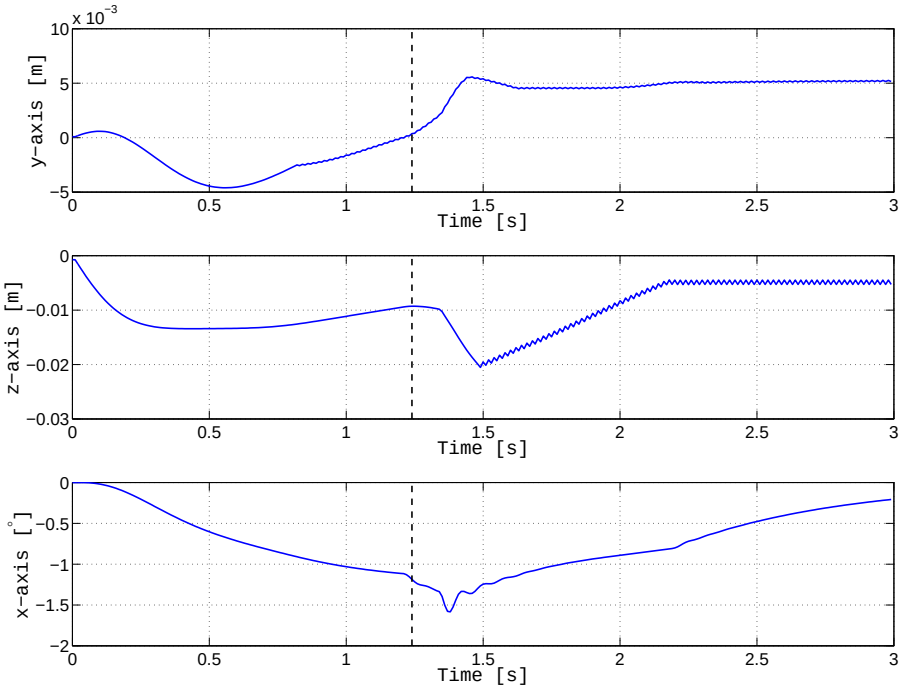


Figure 5-8: Position error of the end-effector for peg-in-the-hole task with compliant robotic arm.

with the vertical line having similar meaning as before. Initial position tracking is good as the stiffness parameters are high. Just before interaction, the arm becomes compliant, adjusts to the goal position error in the command due to its compliance and finally achieves the goal position. Hence, the steady state errors are very small.

The position errors with time are shown in Figure 5-8. They are seen to be considerably low for all three axes. The error along y-axis is mostly due to the command error. the z-axis error is considerably low with a steady-state error in $10e^{-3}$ m. Due to compliant parameters, there will be some steady-state errors. These could further be reduced by making the robot stiff again after the task has been achieved and the robot is stationary. The orientation error about x-axis too is 0.2 deg at steady state.

5-4 Single Object Manipulation Task

This section describes the results of the planning and execution of a manipulation task. The task chosen involved the use of both the robotic arms separately and cooperatively along with the use of transition controllers. The planning involves generation of all contact modes followed by the contact map. Using spatial relationships, transition were added to the Map. The shortest path for on the contact map was determined and object path-planning for each sub-task was done. This was followed by controller assignment and execution. Figure 5-9 displays a set time-lapsed images of the task being executed in VREP. The task conditions are described as follows:

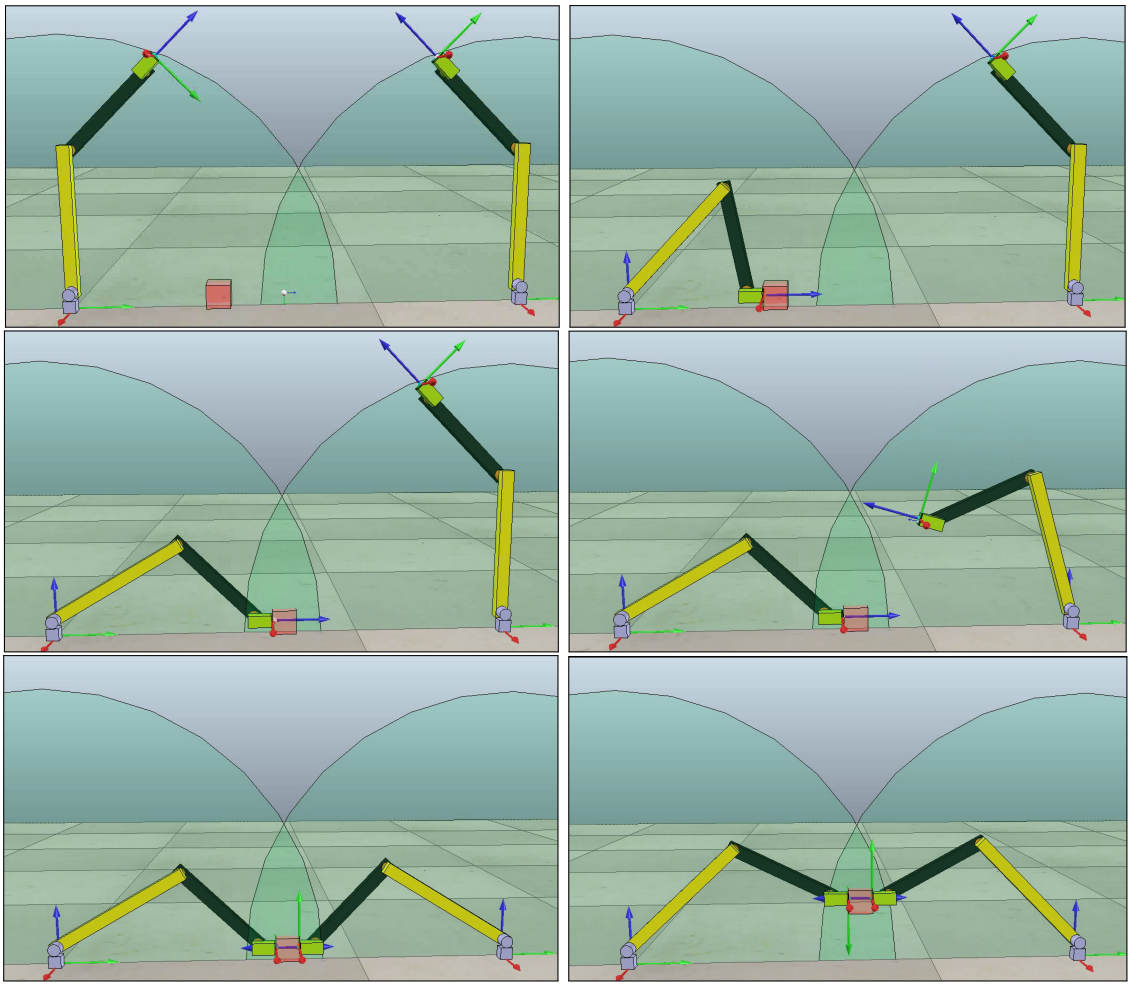


Figure 5-9: Time-lapse photos of the simulated manipulation task in VREP as described in section 5-4 (Left to right, top to bottom).

- The scene consisted of two robots $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$, one object $\boldsymbol{O_1}$ and one surface $\boldsymbol{S_1}$. The robots were controlled via the previously described compliance controllers. Every controller had fixed parameter during execution. The object chosen was cubical in shape. The surface was the ground on which the robots and the object rested.

- The initial and goal positions of the object (centre of mass) were set as:

$$\boldsymbol{p}^{\mathbf{o}}_{\mathbf{init}} = \begin{bmatrix} 0 & 0.24 & 0.02 \end{bmatrix}$$
$$\boldsymbol{p}^{\mathbf{o}}_{\mathbf{goal}} = \begin{bmatrix} 0 & 0.36 & 0.12 \end{bmatrix}.$$

  The initial position of the object was on the ground which is given by the surface $z = 0$. The offset of 0.02 in the $\boldsymbol{p}^{\mathbf{o}}_{\mathbf{init}}$'s y position is because the object is not point mass.

- $\boldsymbol{p}^{\mathbf{o}}_{\mathbf{init}}$ lay only in the workspace of $\boldsymbol{R_1}$ and $\boldsymbol{p}^{\mathbf{o}}_{\mathbf{goal}}$ lay in the common workspace of both the robots. As the robots have no grippers, one single robot cannot lift the object from the ground. Such a task requires two robots manipulating cooperatively.

- Single arm manipulation is limited to a robot pushing the object in the desired direction. The workspace of single arm manipulation is hence, restricted to the surfaces.

The manipulation planning begins with the generation of all contact modes, followed by the contact map for the given manipulation task. This has been described in the next subsection.

5-4-1 Generation of Contact Map

The scene contains two robots $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$, a surface $\boldsymbol{S_1}$ and the object being manipulated $\boldsymbol{O_1}$ (a total of four). Using RCL, the first step is to generate all possible contact modes. Using Equation (3-5), the total number of contact modes are seven. These have been shown in Figure 5-10. Again, from the given condition and assumptions, several of these modes could be omitted.

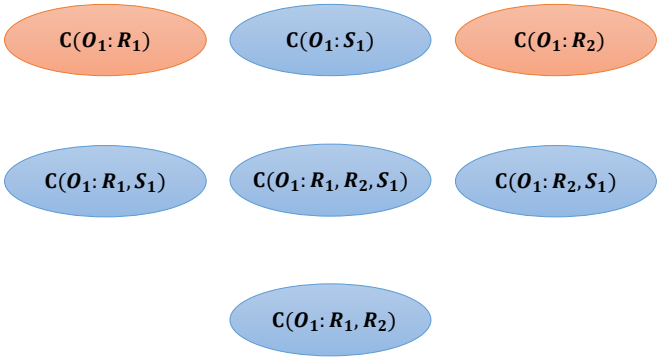


Figure 5-10: $2^{(nt-1)}-1$ combinations of contact modes for the manipulation task in Figure 5-9, where $nt = 4$.

Both the robots have no grippers. Thus, a single arm is only capable of manipulating an object resting on the surfaces within its workspace. It can only push the object. From this assumption, the contact modes marked in red in Figure 5-10 can be neglected. That leaves a total of five. From the given geometrical information of the scene components, spatial relationships amongst the contact modes have been defined and the contact map is generated as shown in Figure 5-11. Using this map, the manipulation task can be divided into sub-tasks.

The initial object position $\boldsymbol{p^o_{init}}$ lies in the contact mode $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$ and the goal position $\boldsymbol{p^o_{goal}}$ in $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2})$. The next step is to find the shortest path on the contact map from the initial to the goal contact mode. This is done via Dijkstra's shortest path algorithm as described in subsection 3-4. The shortest path is shown in Figure 5-12 from the green nodes. They are listed as follows:

- $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$
- $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$
- $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2}, \boldsymbol{S_1})$
- $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2})$

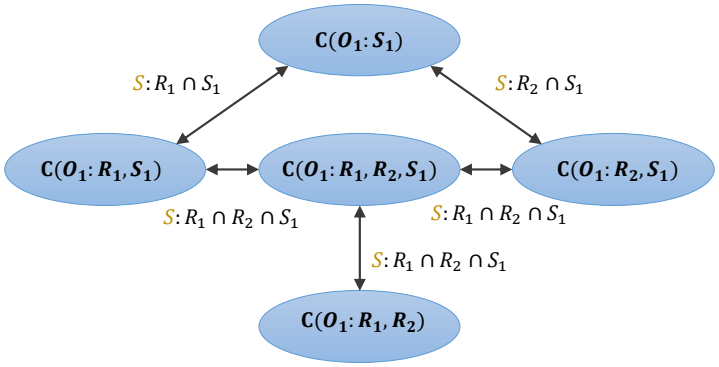


Figure 5-11: The Contact map for the given scene components for the task in Figure 5-9.

These nodes are the relevant contact modes which the object has to pass through, in order to be manipulated. Path-planing and controller assignment needs to be undertaken in each of these contact modes. This is the basic idea of dividing a larger manipulation task into small, multiple tasks. Each of these sub-manipulation task will bring the object closer to its global goal-set. The next subsection describes the object path-planning for the manipulation task.

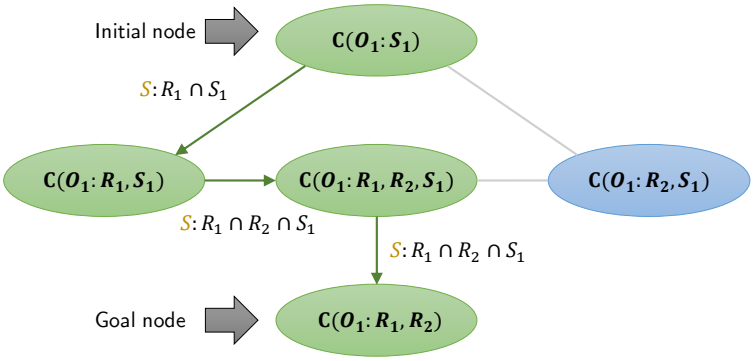


Figure 5-12: Shortest path on contact map for the manipulation task in Figure 5-9 found using Dijkstra's shortest path algorithm, represented as the green nodes.

5-4-2 Object Path-planning

In each of the relevant contact modes, the robot(s) has to manipulate the the object to bring it closer to the global goal position $\boldsymbol{p}^{\mathbf{o}}_{\mathbf{goal}}$ as well as in the workspace of the next contact mode. Planning is done in each of these modes until the one with $\boldsymbol{p}^{\mathbf{o}}_{\mathbf{goal}}$ is reached. The goal position of each contact mode path-planning serves as the initial position of the next one in each relevant contact mode. The overall path is hence, divided into various sub-paths, one in each contact mode. This has been described previously in section 4-2. The path-planning is done via modified Dijkstra's algorithm in the author's research.

The first of the four contact modes is $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{S_1})$. No manipulation can take place in this state as it is a static state. Hence, planner has to move on to the next one, which is $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$.

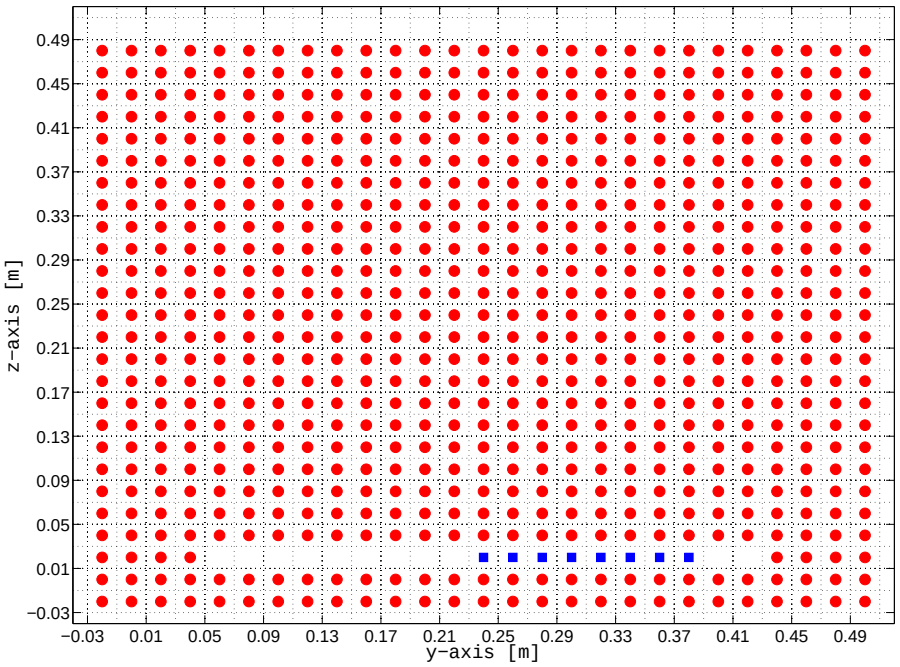


Figure 5-13: Object path for contact mode $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$ of the manipulation task in Figure 5-9. The blue boxes represent the object path and the red dots are obstacles.

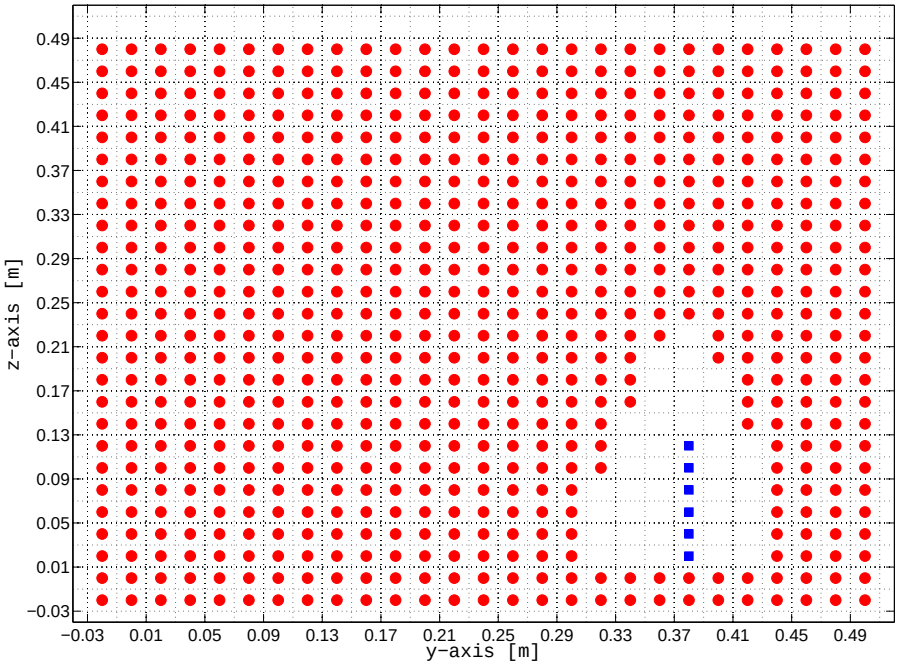


Figure 5-14: Object path for contact mode $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2})$ of the manipulation task in Figure 5-9. The blue boxes represent the object path and the red dots are obstacles.

In this mode, $\boldsymbol{O_1}$ is in contact with $\boldsymbol{R_1}$ and $\boldsymbol{S_1}$. The workspace is given by $\boldsymbol{R_1} \cap \boldsymbol{S_1}$. The remaining workspace is considered to be in the obstacle or non-attainable region. The path generated by the algorithm for this Contact mode is given in Figure 5-13.

In this figure, the blue boxes represent the planned path of the object and the red dots represent the obstacle region (or nodes.) The non-red region represents the workspace of $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{S_1})$. Starting from the initial position $\boldsymbol{p}^{\mathbf{o}}_{\mathbf{init}}$, the goal position in this contact mode is at [0 0.38 0.02]. This position will serve as the initial for next contact mode path-planning.

The next contact mode is $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2}, \boldsymbol{S_1})$, with the workspace $\boldsymbol{R_1} \cap \boldsymbol{R_2} \cap \boldsymbol{S_1}$. It can be observed, both intuitively and numerically, that the initial position is already the best possible position in this contact mode which is closest to the global goal and also lies in the workspace of the next contact mode. Hence, no new path is produced and it is possible to directly move on to the next contact mode, which is $\mathbf{C}(\boldsymbol{O_1} : \boldsymbol{R_1}, \boldsymbol{R_2})$.

In this contact mode, there are two robots and no surfaces, and thus it is possible to cooperatively manipulate the object in the common robot workspace, given by $\boldsymbol{R_1} \cap \boldsymbol{R_2}$. This also contains the global goal position of the object, as it is also the goal node in the contact map shortest path. This, path-planning will end here. The object manipulation path is shown is Figure 5-14. The final object position coincides with $\boldsymbol{p}^{\mathbf{o}}_{\mathbf{goal}}$.

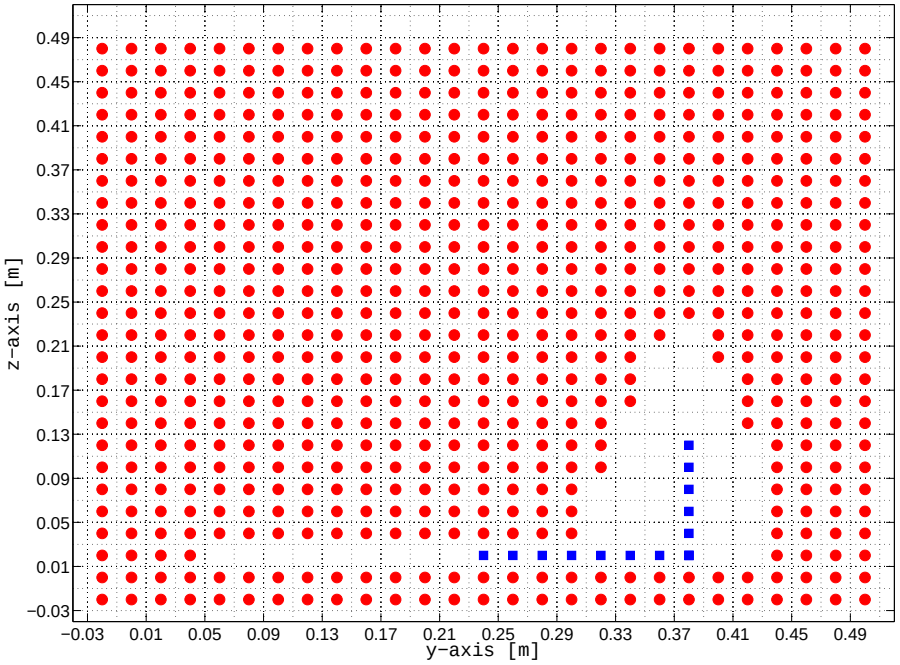


Figure 5-15: Object path for the complete manipulation task of Figure 5-9. The blue boxes represent the object path and the red dots are obstacles.

The two paths shown in Figure 5-13 and 5-14 together make up the overall path of the object to be manipulated. This is shown in Figure 5-15. This is the desired path the object needs to follow to successfully reach from the given initial and goal positions. The next step is to assign controllers to track each of these generated local paths.

5-4-3 Controller Assignment

The controller assignment is done based on the flowchart given in Figure 4-4. There are two manipulation sub-paths in total for the given manipulation task. These would require two manipulation controllers. The robots also need to make or break contact with the object being manipulated, which would require transition controllers. These controllers have been described in section 4-4.

The first controller needed will be $\boldsymbol{R_1}$ making contact with $\boldsymbol{O_1}$, which requires a make-contact controller. The contact point will be a point on the surface of $\boldsymbol{O_1}$. Once the contact is made, $\boldsymbol{R_1}$ needs to perform a manipulation task of pushing the object according the object path. Hence, a single arm manipulation controllers is assigned. The next manipulation task is cooperative manipulation task of lifting the object. Before that is possible, $\boldsymbol{R_2}$ needs to make contact with the object, which is done by assigning another Make-Contact controller. Controllers are assigned to both $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ for the cooperative manipulation that follows, for the task of lifting up the box.

On having assigned the controllers needed for the manipulation task, dynamical information is used to define the DoA and assign a goal-set to each. Switching condition are then established and the controllers are sequentially composed and ready to be executed in a hybrid fashion. The results of the execution of the hybrid automaton have been discussed in the subsection that follows.

5-4-4 VREP Simulation Results

The hybrid automaton for the given task has been executed in VREP. This subsection discusses the various results obtained. The positions, forces and velocities of the two robotic arms and the object being manipulated have been discussed and analysed. The complete task execution took around 30 seconds in total. The controller switching are demarcated by vertical, dashed lines on every graph. There are three of these lines which divide the graph into four sections, one for each controller.

Comparison of Position Errors

Figure 5-16 shows the position tracking of the first robot, $\boldsymbol{R_1}$ (left robot in VREP). The red line represents the desired or virtual end-effector position and orientation and the blue one is the actual end-effector ones. The manipulation task starts with $\boldsymbol{R_1}$ making contact with $\boldsymbol{O_1}$, following by it pushing the object. This can be seen by the first and second sections of the graphs. The end-effector maintains good accuracy during position tracking. In the third section, $\boldsymbol{R_1}$ remains stationary and hence the lack of movement in the graphs.

The third section represents the make-contact controller of the second robot , $\boldsymbol{R_2}$ (right robot in VREP), with the object. Figure 5-17 shows the position tracking of $\boldsymbol{R_2}$. It has no activity in the first two sections as $\boldsymbol{R_1}$ is active then. The third graph section shows $\boldsymbol{R_2}$ positioning itself to make contact with $\boldsymbol{O_1}$. Again, the end-effector maintains a good tracking accuracy.

In the last section of the graph, the two arms are cooperatively manipulating to lift the object, as seen from the z direction movement in the graph. The y direction for both the arms has

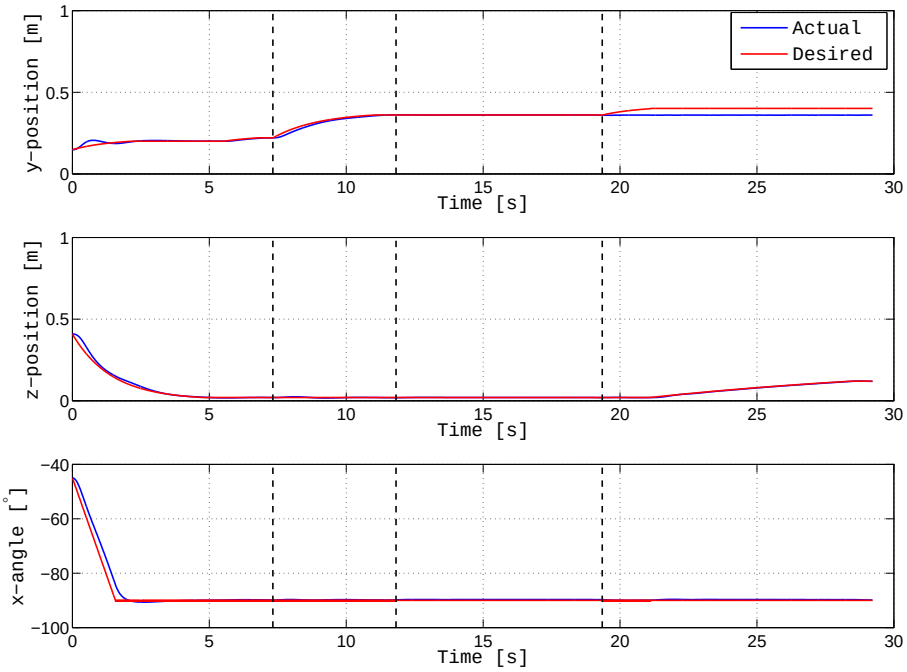


Figure 5-16: Position tracking of R_1 for the simulated manipulation task.

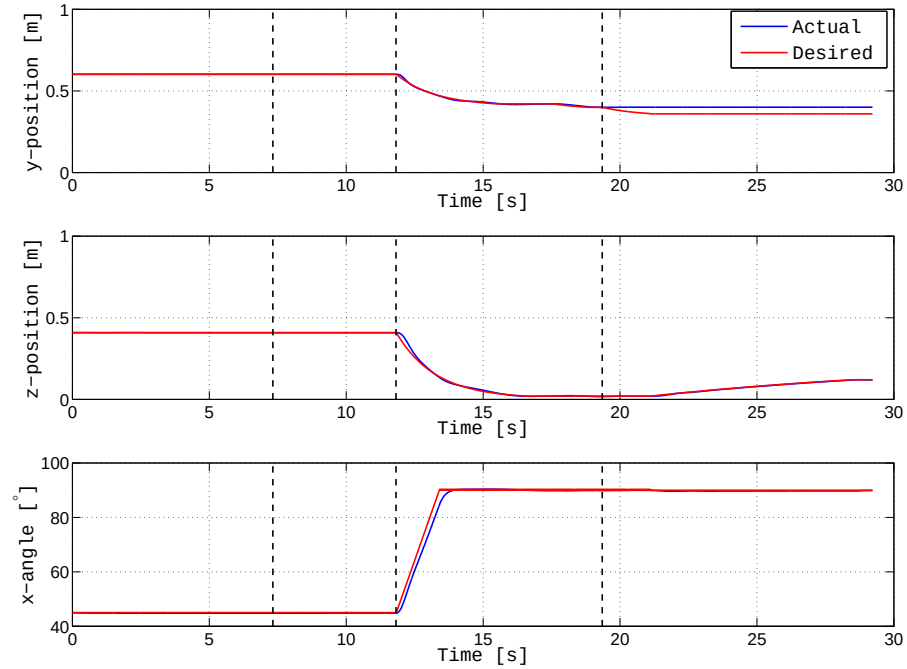


Figure 5-17: Position tracking of R_2 for the simulated manipulation task.

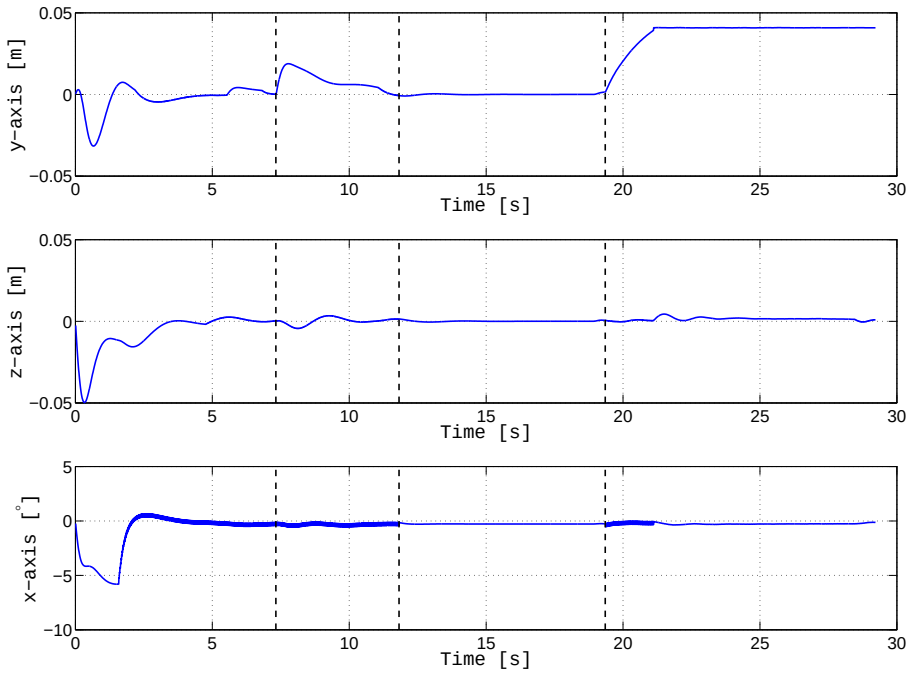


Figure 5-18: Position errors of $\boldsymbol{R_1}$ for the simulated manipulation task.

a constant steady state error. This is because the desired end-effector position is somewhere "inside" $\boldsymbol{O_1}$ and can never be reached as the motion is constrained by the object. Instead, it leads to higher grasping forces due the difference between the desired and actual end-effector positions. This has been discussed previously, in section 2-6.

The orientation tracking for both arms is also seen to be performing well. The errors of position tracking are shown in Figure 5-18 and 5-19 for robots $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ respectively. The tracking error can be seen not exceeding a very high value, with the steady state error asymptotically reaching zero when the desired positions are constant and the arms are not constrained.

The position and orientation data of the object with time was also measured and is shown in Figure 5-20. As only the second and fourth controllers are meant for manipulation, the movement of $\boldsymbol{O_1}$ is only in those two sections of the graph. During the single arm manipulation task, $\boldsymbol{O_1}$ is only pushed in the positive y direction and during cooperative manipulation, it is lifted by the two robots in the positive z direction, as seen from the figure. The orientation remains constant with some jitters. The comparison of the desired and actual positions of $\boldsymbol{O_1}$ in the yz plane is displayed in Figure 5-21. It can be observed that the desired object path is correctly tracked by $\boldsymbol{O_1}$ during manipulation.

Comparison of Velocities

The velocity profiles of the two robotic arms $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ for the manipulation task are given in Figure 5-22 and 5-23 respectively. The velocities have an under-damped response for both

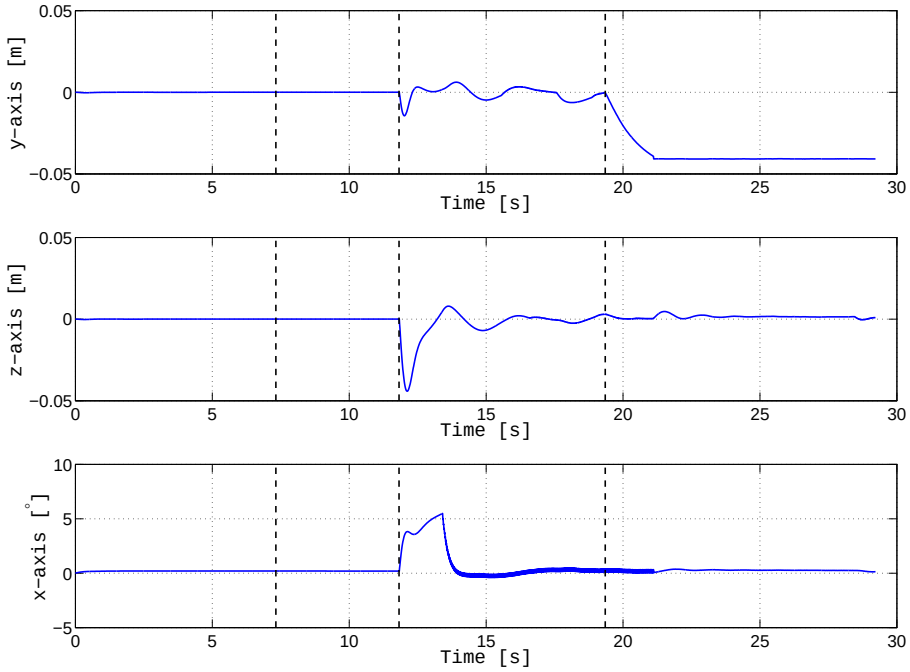


Figure 5-19: Position errors of R_2 for the simulated manipulation task.

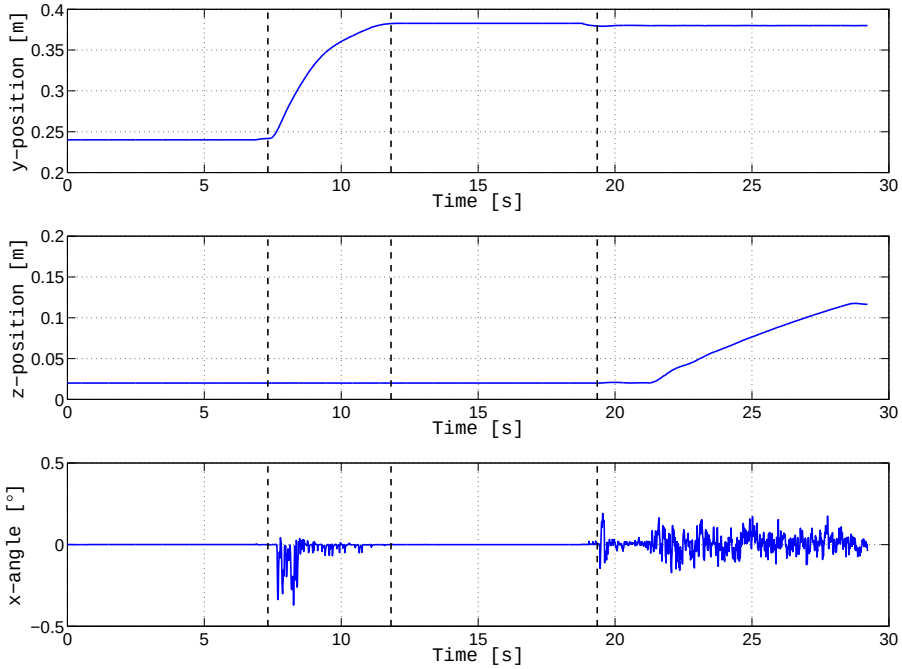


Figure 5-20: Position and orientation of O_1 for the simulated manipulation task.

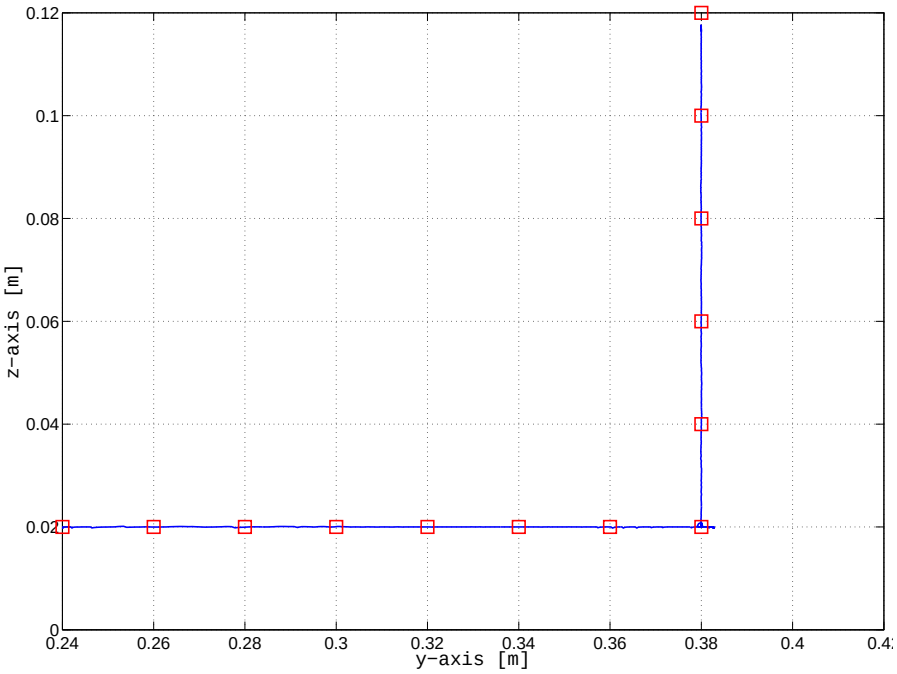


Figure 5-21: Actual and desired positions of $\boldsymbol{O_1}$ for the simulated manipulation task. The red boxes represent the desired object position from path-planning and the blue line represents the path taken by $\boldsymbol{O_1}$ in the yz plane.

the arms. This could be the high acceleration or the sudden change in the desired end-effector positions. An improvement in the trajectory generation of the desired end-effector position of the robotic arms could result in smoother velocity profiles. Avoiding sudden jumps (accelerations and jerks) in the desired velocities could also lead to a smoother response of the robotic arm and consequently, making the manipulation task safer, smoother and more conducive for fragile objects.

The velocity in each direction is close to zero when there is a switching of controllers. This can be observed by checking the velocities where the dashed vertical lines intersect the graphs. This is due to the fact that the goal-set of each robot has the velocity states equal to zero. It is only possible to switch to the next controller when the previous controller reaches its goal-set.

Comparison of Forces

Forces at the end-effector represent the robotic arm in contact. Figure 5-24 and Figure 5-25 represent the end-effector forces acting on $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ respectively. It can be observed that the forces are only present when there is contact of the robotic arm with the object, that is, during the execution of manipulation controller and while making the final contact.

In both the figures, the forces in the y-axis and torques about x-axis do not have a significant value. This is because the forces during contact are mainly acting in direction normal to the end-effector (or object) contact surface, which is the z-axis with respect to the end-effector.

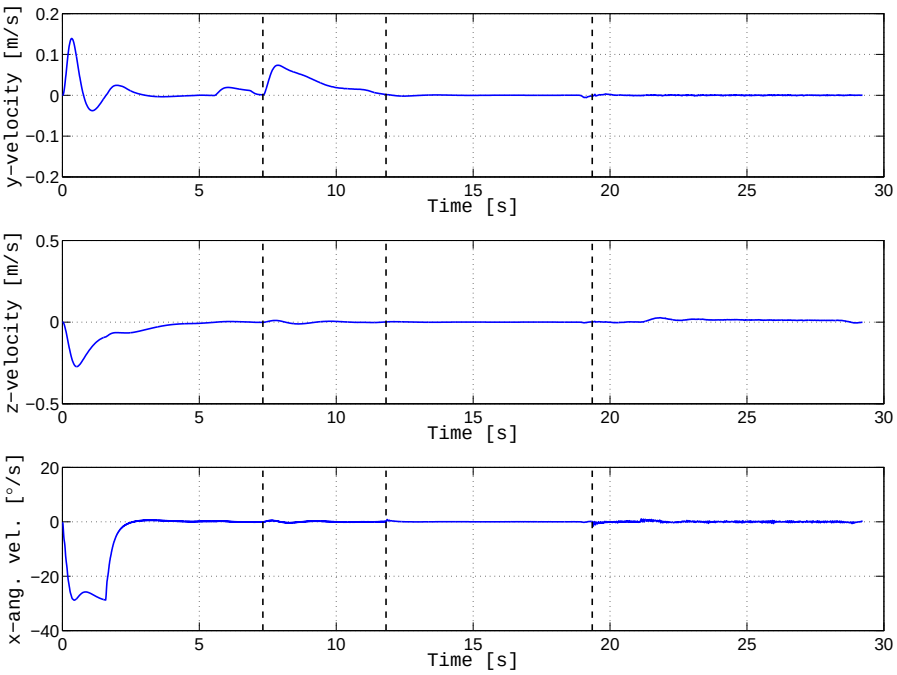


Figure 5-22: Velocities of R_1 for the simulated manipulation task.

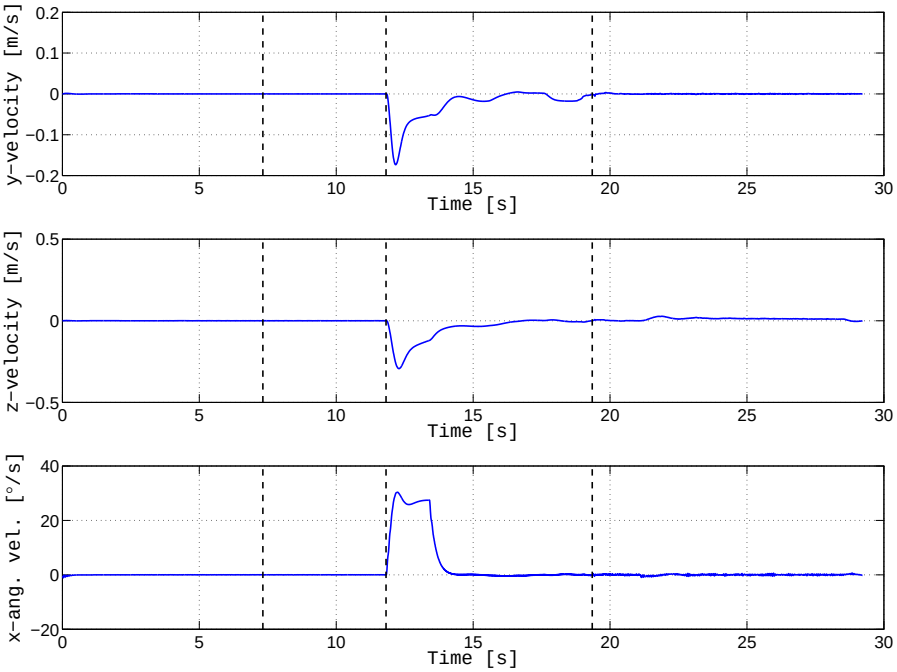


Figure 5-23: Velocities of R_2 for the simulated manipulation task.

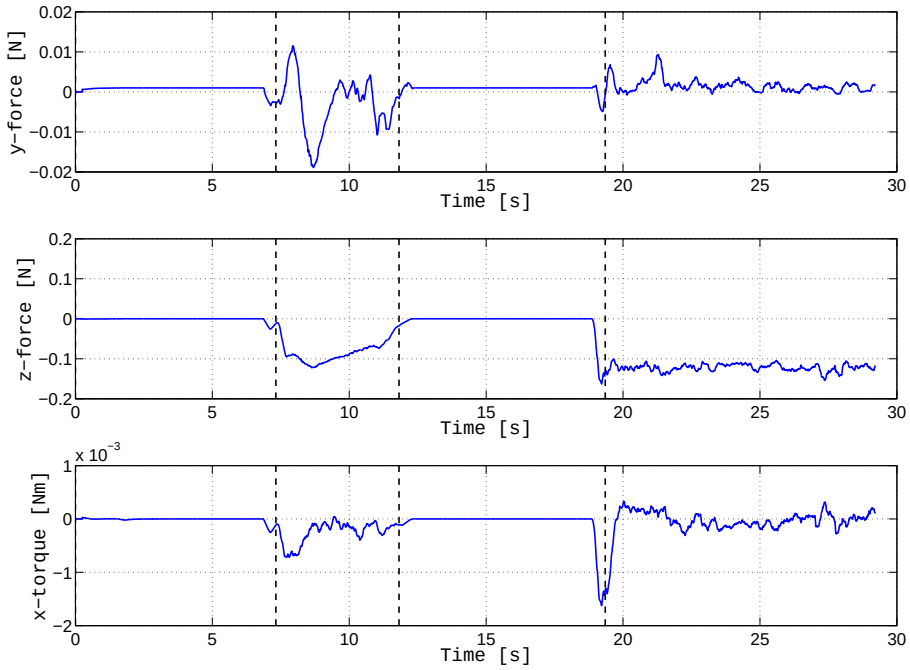


Figure 5-24: End-effector forces of R_1 for the simulated manipulation task.

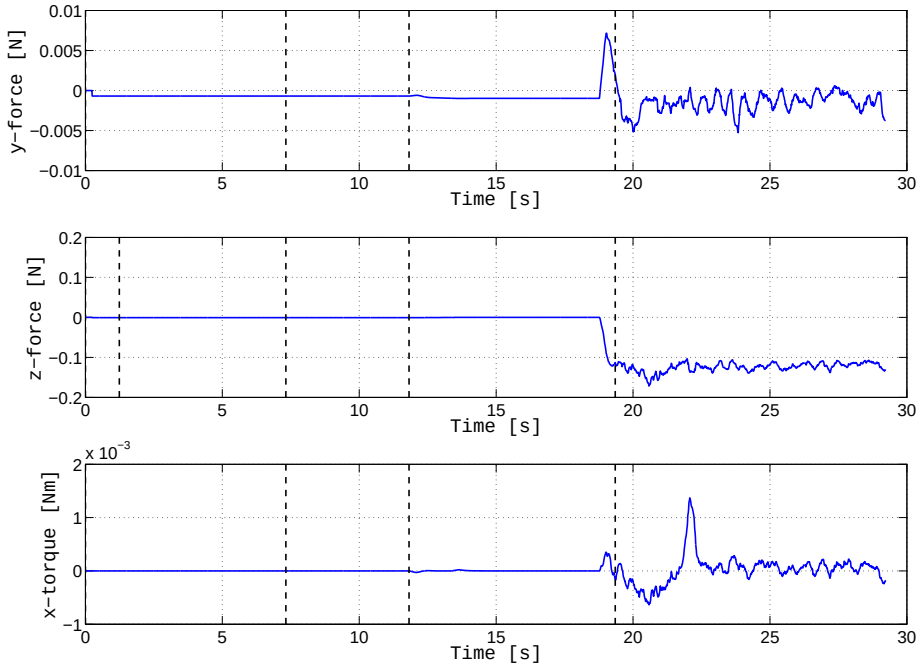


Figure 5-25: End-effector forces of R_2 for the simulated manipulation task.

The manipulation controllers controller assigned were compliant to an extent of not generating excessive end-effector forces while maintaining a considerable position accuracy.

When there is a switch from a non-contact controller to manipulation controllers, there are forces always present at the robot end-effector along the z-axis. This can be observed by checking the points at which the vertical, dashed lines intersect the horizontal axis. The presence of the z-axis forces represent an establishment of contact of the robot with $\boldsymbol{O_1}$. Unless contact has not been established, the robot cannot switch to a manipulation controller. This is a part of the switching condition in the hybrid automaton. This can be observed for $\boldsymbol{R_1}$ while switching to the push controller and for both the arms while switching to cooperative manipulation controller.

5-5 Multiple Object Manipulation Task

Multiple object manipulation is done similar to the single object manipulation except that a correct manipulation sequence might be necessary due to co-dependencies among the objects. The sequence is of manipulation is found via Algorithm 1 described in section 3-5-1. The input to the algorithm is the initial and goal positions of the objects to be manipulated, along the geometrical information of the scene. The algorithm returns the correct sequence of manipulating the set of objects.

For this purpose, a small case study was done involving 4 objects, which could be named as $\boldsymbol{O_1}$, $\boldsymbol{O_2}$, $\boldsymbol{O_3}$ and $\boldsymbol{O_4}$. Different initial and final positions were implemented and the algorithm was tested in order to produce the manipulation sequences in each case. To simulate static dependencies of the objects with each other, a task of stacking the objects was studied. For simplicity, all objects were of similar geometry. They were cubic in shape with a length of 0.02 meters. The scene also comprised of a ground surface $\boldsymbol{S_1}$ and two robots $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$. The results have been presented in Table 5-1.

Table 5-1: Sequence of object manipulation with varying initial and goal object positions with object dependencies. The positions are given in meters with respect to base frame.

$\boldsymbol{O_1}$		$\boldsymbol{O_2}$		$\boldsymbol{O_3}$		Sequence
init	goal	init	goal	init	goal	
[0.26 0.02]	[0.36 0.02]	[0.24 0.02]	[0.36 0.10]	[0.56 0.02]	[0.36 0.06]	1, 3, 2
[0.24 0.02]	[0.36 0.10]	[0.24 0.02]	[0.36 0.02]	[0.56 0.02]	[0.36 0.06]	2, 3, 1
[0.24 0.02]	[0.36 0.06]	[0.56 0.02]	[0.36 0.10]	[0.50 0.02]	[0.36 0.02]	3, 1, 2
[0.24 0.02]	[0.32 0.06]	[0.28 0.02]	[0.32 0.02]	[0.50 0.02]	[0.42 0.02]	2, 1, 3

On find the correct sequence of object manipulation, the similar concepts of single object manipulation from the previous section can be applied to manipulate the object one after another, via their contact maps. Other objects could be considered as obstacles when they not being manipulated. It is also possible that two objects can be manipulated simultaneously if there are no dependencies and there are robots available for each manipulation. This could be a part an extension of the author's work.

5-6 Summary

This chapter presents the important results of the simulations performed during the research work. First, the results of two arms performing the peg-in-the-hole task have been compared, where one arm is stiff and other is comparatively compliant. The compliant arm was seen to achieve the task in the presence of a nominal error in the constrained plane whereas the stiff robotic arm gets stalled along with high interaction forces.

This result is followed by a detailed analysis of the simulation results of the planning and execution of a complete task of manipulating an object with two robotic arms in VREP. The end-effector positions, forces and velocities are analysed followed by object's positions and velocities. Manipulation planning was performed via Robot Contact Language and contact maps and the controllers were executed in VREP and Matlab to successfully execute the manipulation task. Tracking errors and forces for both the arms and the object were observed to be within acceptable bounds.

Lastly, the simulation results of Algorithm 1 is compared with a scene of three objects with static dependencies. The object manipulation sequences provided by the algorithm were seen to be appropriate and consistent for varying initial and goal positions of the objects being manipulated.

Chapter 6

Conclusions and Future Work

6-1 Conclusions

This chapter draws conclusions about various theories and results presented in the author's research. The main idea proposed in this research work was the formulation and usage of Robot Contact Language (RCL). This idea was used to simulate compliant robotic arms to perform a complex task with multiple controllers. The idea of sequential compositions was used to execute the controllers in a hybrid manner and perform the manipulation task.

RCL uses combinatorics along with spatial relationships to divide a complex manipulation task into various sub-tasks based on the contact of the manipulated object with other objects, robots and surfaces. Each contact mode is a combination of two or more components present in a given scene. Using some basic rules to make and break contact amongst these modes, a map, called the contact map, can be build on a symbolic level. This map already gives some insight to carrying out a manipulation task on a higher level.

If an object has to be picked and placed from one surface to another using a robot, it involves the robot making contact with the object, breaking its contact with the first surface, making contact with the other surface and finally breaking its own contact with the object. This information is already embedded in the contact maps without the use of any geometric information. This information is used in the next step, where more information is added to the contact map for a specific manipulation planning.

The Geometric information consists of robot workspaces, the object shapes, size, grasps, etc., the contact areas of objects and surfaces, the space occupied by obstacles and other such relevant information. All these quantities can be expressed using basic geometrical entities. This information is used to make the contact maps richer and more useful. Spatial relationships are defined amongst the contact modes and it is hence possible to define a workspace of each mode along with the workspace for the transition between two modes.

Given the initial and final positions of an object to be manipulated, the contact map can be then used to plan a complex manipulation task by separating the task into various sub-tasks

with one for each relevant contact mode. The modes containing the initial and final object positions can be identified and the shortest path on the contact map can be found between the two contact modes. This can be done using a graph search algorithm. In the author's research the Dijkstra's shortest path algorithm has been used.

On having found the shortest path on the contact map between the initial and final contact modes, the manipulation task can be divided into smaller tasks, one for each contact mode present in the shortest path (if necessary). Manipulation path-planning can now be done separately and sequentially in each of these relevant contact modes with the aim to bring the object being manipulated closer to the global goal position. With overlapping workspaces, path-planning in each contact mode will lead to a connected global path for the manipulation task. A controller can be assigned to each of these sub-manipulations to carry out their respective task, along with transition controllers to make or break the contact of robot with the manipulated object.

This is followed by using dynamical information to decipher the domains of attraction (DoAs) and assign goal-sets for each controller. The switching conditions amongst these controllers is derived and the controllers are sequentially composed and executed in a hybrid manner to carry out the manipulation task. Various simulations were performed on the Virtual Robotics Experimentation Platform (VREP) and Matlab software and results were analysed.

The controller used by the robotic arms for manipulating objects was spatial springs based impedance controller. As interaction of robots with the environment necessitates some form of force control, this particular compliance controller was used to carry out manipulation tasks as well as position control of the robot, which requires no interaction with the environment. This controller was found to have a number of advantages over conventional force controllers and workspace controllers. The main advantages were that it did not require inverse kinematics for workspace control and force control was achieved by varying the spatial spring stiffness parameters and/or the desired end-effector position. The end-effector force was seen to be proportional to the distance between the actual and desired end-effector positions.

Two robotic arms were simulated to perform the task of peg-in-the-hole. One arm was stiff and other was made compliant. The resultant interaction forces and position accuracies were analysed. The compliant arm was seen to perform the task with very low and short lived interaction forces even with errors in the goal (hole) position. Similar conditions were simulated for the stiff arm which was unable to accomplish the task successfully and got stuck on the outer surface of the hole with very high interaction forces. The compliant robot arm showed the property to adapt to position errors and "wiggle" around or about a surface to accomplish the task. On the other hand, the stiff arm was seen trying to accurately track the goal position and in the end, failing to perform a task which involved interaction with the external environment.

To study the planning and execution of a complex manipulation task, a task involving various controllers like making contact, pushing and cooperative manipulation, was chosen. A step-wise planning was done, starting from the generation of contact modes and the contact map. It was seen that the use of RCL indeed breaks down the manipulation task into smaller and easier-to-plan sub-tasks. As stated previously, path-planning and assignment was done and the controllers were sequentially composed and the control automaton was executed. Various results of the task variables such as the force, position and velocities of the arms and the object being manipulated were analysed.

The manipulation task was successfully performed. The forces acting on the object during single arm and cooperative manipulation did not exceed a very high value by choosing the correct stiffness values. The grasping forces could further be set by varying the desired end-effector positions. Cooperative manipulation task was seen to satisfy the grasp constrains, which lead to successful cooperative manipulation and avoided the object from losing the grasping forces. The positions errors throughout the manipulation tasks were within acceptable bounds throughout the task. The actual path of the object being manipulated complied with the desired manipulation path and the goal object was achieved at the end of the manipulation.

The velocities were also within reasonable bounds. A few jerks were seen in the initial part of the manipulation. This was due to sudden change in the desired position and velocity of the end-effector. These could be reduced by further improving the trajectory generation algorithm.

The concept of RCL was also extended to the manipulation of multiple objects in a scene. A global contact map was derived which involved all the objects which were required to be manipulated along with the other components in the scene. Using that map, it was shown how any multiple manipulation task could be achieved at an abstract level with just the combinatory map. If there are dependencies of an object being manipulated with other objects in the scene, those objects were manipulated first.

This lead to the idea of developing an algorithm that resolves contact constraints or dependencies in a multiple object manipulation scenario. Given the initial and goal positions of all the objects to be manipulated, the algorithm uses the contact maps to calculate object dependencies and return the correct sequence of objects to be manipulated. The algorithm was validated by checking its performance with various different object initial and goal positions and obtaining the results. It was found to be working accurately for all the given positions for a simple manipulation task.

6-2 Future Work

The work undertaken in this M.Sc. research involves the knowledge from a varied robotics fields. It hence, opens up the possibility of exploring these research directions to further improve the concepts developed in the author's research work, some of which will be discussed in brief. Figure 6-1 displays the important fields which could be explored by a future researcher who is interested to extend the idea presented in this book.

The idea of Robot Contact Language is fairly new. The nomenclature has further scope of improvement along with its extension to multiple objects and complex manipulation scenarios and parallel manipulations. Smarter algorithms for high level decision making could be developed with a proper integration, which could automate the complete planning and execution process to the greatest extent.

This concept heavily depends on the availability of geometrical information and the all the components of the scene. Simple robot workspaces, object shapes and surfaces have been simulated in this research work. Extending RCL to complex scenarios, where there are multiple objects with vary geometrical features and non-smooth, non-planar surfaces poses a great

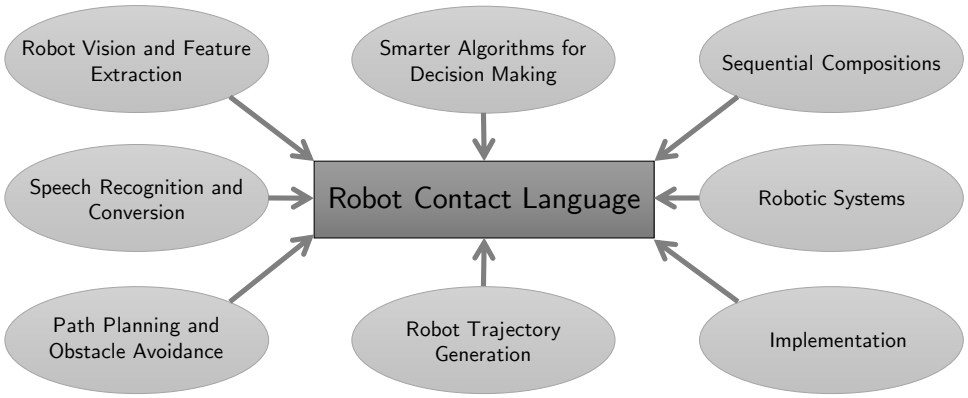


Figure 6-1: Aspects involving potential future work in the field of Robot Contact Language.

challenge. Defining grasp points, contact surfaces, stable object placements, etc. would be more involved in such a case.

Path-planning and trajectory generation algorithms used in this research are quite basic. Dijkstra's algorithm is far from optimal in object path-planning when it is used in 3D Cartesian space as the time complexity increases exponentially while going from 2D to 3D space. Other path planning algorithms based on workspace planning could be researched and adapted to be used in the RCL framework. Same applies for trajectory generation.

Further work also needs to be done with regards to sequential compositions and proper evaluation of controller Domain of Attraction. The idea proposed in this research in more planning based. If a disturbance acts on an object, there is no way to correct it. If object states could be observed or estimated, sequential compositions could be used to their true potential. If a disturbance would lead to a change in contact mode, the supervisory controller could switch to the correct controller and the manipulation task could be resumed by observing the states of the object being manipulated.

This idea has only been evaluated using planar robotic manipulators. It could further be extended to six degrees-of-freedom manipulators and even other non-holonomic robotic systems like UAVs and UGVs. Going one step further, its implementation on a physical experimental setup is also a challenging but important undertaking for the evaluation and analysis of RCL framework. Lastly, speech recognition algorithms could be adapted to convert human speech to RCL which would enable the robot to understand commands in human language.

6-3 Epilogue

The aim of this research work was to bring ideas from varied fields of robotics and control theory and integrate them to devise a robotics framework that would enable higher level decision making in the robot. This was successfully achieved by introducing the Robot Contact Language, development of smarter algorithms and automated execution of the complete framework to perform a simple manipulation task. The idea of compliant robotic manipulation, cooperative manipulation and sequential compositions were successfully integrated and implemented in the framework developed in this research work.

Bibliography

[1] F. Caccavale, P. Chiacchio, A. Marino, and L. Villani, "Six-dof impedance control of dual-arm cooperative manipulators," *IEEE/ASME Transactions on Mechatronics,*, vol. 13, no. 5, pp. 576–586, 2008.

[2] "Baxter Robot, Rethink Robotics ." https://www.youtube.com/watch?v=Mr7U9pQtwq8.

[3] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[4] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*, vol. 3. Wiley New York, 2006.

[5] M. T. Mason, "Compliance and force control for computer controlled manipulators," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 11, no. 6, pp. 418–432, 1981.

[6] J. J. Craig and M. Raibert, "A systematic method of hybrid position/force control of a manipulator," in *The IEEE Computer Society's Third International Conference on Computer Software and Application (COMPSAC)*, pp. 446–451, IEEE, 1979.

[7] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic, "Dual arm manipulation - a survey," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1340–1353, 2012.

[8] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2008.

[9] N. Hogan, "Impedance control: An approach to manipulation," in *American Control Conference, 1984*, pp. 304–313, IEEE, 1984.

[10] N. Hogan, "Impedance control: An approach to manipulation: Part ii - implementation," *Journal of dynamic systems, measurement, and control*, vol. 107, no. 1, pp. 8–16, 1985.

[11] C. Ott, *Cartesian Impedance Control: The Rigid Body Case*. Springer, 2008.

[12] M. A. Diftler, J. Mehling, M. E. Abdallah, N. A. Radford, L. B. Bridgwater, A. M. Sanders, R. S. Askew, D. M. Linn, J. D. Yamokoski, F. Permenter, *et al.*, "Robonaut 2-the first humanoid robot in space," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2178–2183, IEEE, 2011.

[13] C. Fitzgerald, "Developing baxter," in *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pp. 1–6, IEEE, 2013.

[14] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.

[15] E. Najafi, G. A. Lopes, and R. Babuška, "Automatic synthesis of sequential composition of controllers,"

[16] V. Kallem, A. T. Komoroski, and V. Kumar, "Sequential composition for navigating a nonholonomic cart in the presence of obstacles," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1152–1159, 2011.

[17] S. Stramigioli, *From differentiable manifold to interactive robot control*. Delft University of Technology, 1998.

[18] E. D. Fasse and J. F. Broenink, "A spatial impedance controller for robotic manipulation," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 4, pp. 546–556, 1997.

[19] V. Perdereau and M. Drouin, "Hybrid external control for two robot coordinated motion," *Robotica*, vol. 14, no. 02, pp. 141–153, 1996.

[20] M. Uchiyama and P. Dauchez, "A symmetric hybrid position/force control scheme for the coordination of two robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 350–356, IEEE, 1988.

[21] T. Wimbock, C. Ott, and G. Hirzinger, "Impedance behaviors for two-handed manipulation: Design and experiments," in *IEEE International Conference on Robotics and Automation*, pp. 4182–4189, IEEE, 2007.

[22] D. Williams and O. Khatib, "The virtual linkage: A model for internal forces in multi-grasp manipulation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1025–1030, IEEE, 1993.

[23] E. Najafi, G. A. Lopes, and R. Babuska, "Reinforcement learning for sequential composition control," in *IEEE 52nd Annual Conference on Decision and Control (CDC)*, pp. 7265–7270, IEEE, 2013.

[24] R. B. Rusu, N. Blodow, Z. Marton, A. Soos, and M. Beetz, "Towards 3d object maps for autonomous household robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3191–3198, IEEE, 2007.

[25] A. Pronobis and P. Jensfelt, "Large-scale semantic mapping and reasoning with heterogeneous modalities," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3515–3522, IEEE, 2012.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *et al.*, "Dijkstra's algorithm," 2001.

[27] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[28] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *IEEE International Conference on Robotics and Automation*, pp. 3327–3332, IEEE, 2007.

[29] S. M. LaValle, "Rapidly-exploring random trees a Đęew tool for path planning," 1998.

[30] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 995–1001, IEEE, 2000.

[31] M. V. Weghe, D. Ferguson, and S. S. Srinivasa, "Randomized path planning for redundant manipulators without inverse kinematics," in *7th IEEE-RAS International Conference on Humanoid Robots*, pp. 477–482, IEEE, 2007.

[32] R. Alami, J.-P. Laumond, and T. Siméon, "Two manipulation planning algorithms," in *Proceedings of the workshop on Algorithmic foundations of robotics*, pp. 109–125, AK Peters, Ltd., 1995.

[33] C. Robotics, "V-rep, virtual robot experimentation platform," 2013.

Appendix A

Glossary

List of Acronyms

SCC	sequential composition control
P(I)D	proportional-(integral)-derivative
IK	inverse kinematics
DoA	domain of attraction
GCM	global contact map
LCM	local contact map
RCL	Robot Contact Language
RCC	Resolve Contact Constraints
RSC	Resolve Spatial Constraints
DoF	degree-of-freedom

List of Symbols

$\boldsymbol{q}$	generalized/joint position vector
Q	joint space
$\boldsymbol{\Psi_0}$	base(inertial) frame
$\boldsymbol{\Psi_n}$	n^{th} link frame
$\boldsymbol{H}$	Homogeneous matrix
$\boldsymbol{R}$	Rotation matrix
$\boldsymbol{p}$	translational position vector
$\boldsymbol{T}$	Twist
$\boldsymbol{w}$	angular velocity
$\boldsymbol{v}$	translational velocity
$\boldsymbol{W}$	Wrench
$\boldsymbol{m}$	torque vector
$\boldsymbol{f}$	force vector
$\boldsymbol{J}$	Jacobian matrix
$\boldsymbol{M}(\boldsymbol{q})$	Manipulator inertia matrix
$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$	Coriolis matrix
$\boldsymbol{G}(\boldsymbol{q})$	Gravity matrix
$\boldsymbol{\tau}$	Joint torque co-vector
$\boldsymbol{W}_{\text{ext}}$	External end-effector Wrench
$\boldsymbol{K}_{\text{d}}$	Derivative gain
$\boldsymbol{K}_{\text{p}}$	Proportional gain
$\boldsymbol{q}_{\text{d}}$	Desired joint position vector
F_{spg}	One-dimensional spring force
x	Actual spring position
x_{d}	desired/rest-length spring position
K_{spg}	Spring stiffness
$\boldsymbol{\Lambda}_{\text{d}}$	Desired end-effector inertia
$\boldsymbol{D}_{\text{d}}$	Desired end-effector damping
$\boldsymbol{K}_{\text{d}}$	desired end-effector stiffness
$\boldsymbol{W}_{\text{ee}}$	Generated end-effector Wrench
$\tilde{\boldsymbol{x}}$	Infinitesimal end-effector Twist
E_{spg}	Energy in 1D spring
$\boldsymbol{K}_{\text{t}}$	Translational stiffness
$\boldsymbol{K}_{\text{r}}$	Rotational stiffness
$\boldsymbol{K}_{\text{c}}$	Coupling stiffness
$\boldsymbol{G}_{\text{t}}$	Translational co-stiffness
$\boldsymbol{G}_{\text{r}}$	Rotational co-stiffness
$\boldsymbol{G}_{\text{c}}$	Coupling co-stiffness
$\boldsymbol{W}_{\mathbf{f}}$	Generated end-effector Wrench via spatial springs

$\boldsymbol{D_c}$	Cartesian damping matrix
$\boldsymbol{W^o_{ext}}$	Object external forces
$\boldsymbol{W^o_{int}}$	Object internal forces
$\boldsymbol{A}$	Grasp matrix
$\boldsymbol{B}$	Null-space matrix of grasp matrix
$\boldsymbol{T_o}$	Object trajectory
$\boldsymbol{T_k}$	k^{th} arm trajectory
$\boldsymbol{x}$	State vector of a system
$\boldsymbol{u}$	Input vector of a system
$\mathcal{X}$	State space
$\mathcal{U}$	Input space
Φ	Generic controller/control law
$\mathcal{D}$	Domain of Attraction (DoA)
$\mathcal{G}$	Goal-set of a controller
$\boldsymbol{x}_0$	Initial state vector
Φ_{up}	Up controller
Φ_{down}	Down controller
Φ_{swing}	Swing-up controller
θ_{p}	Pendulum angular position
$\boldsymbol{f}_{\text{ext}}$	External force vector
S	State space
$\boldsymbol{c}$	arbitrary constant
$\boldsymbol{O_x}$	Object "x" in the scene
$\boldsymbol{R_x}$	Robot "x" in the scene
$\boldsymbol{S_x}$	Surface "x" in the scene
nr	Number of robots
no	Number of object
ns	Number of surfaces
nc	Total combinations of contact combination / contact modes
nt	Total number of scene components
d_0	Total length of a stretched planar robotic arm
$\boldsymbol{\mathcal{W}_R}$	Robot workspace
$\boldsymbol{\mathcal{W}_S}$	Contact plane/workspace of a surface
$\boldsymbol{\mathcal{W}_{obs}}$	Total obstacle workspace
$\mathcal{GE}$	scene geometry (Algorithm 1)
$\boldsymbol{x^i_{1:no}}$	initial object positions (Algorithm 1)
$\boldsymbol{x^g_{1:no}}$	goal object positions (Algorithm 1)
$\boldsymbol{LS}$	Local object sequence (Algorithm 1)
$\boldsymbol{GS}$	Global object sequence (Algorithm 1)
$\boldsymbol{CMap}$	Contact Map (Algorithm 1)
$\boldsymbol{minP}$	Shortest path on Contact Map (Algorithm 1)
$\boldsymbol{DO}$	Set of dependable objects (Algorithm 1)
$\boldsymbol{x}_{\text{init}}$	initial position (of an object)
$\boldsymbol{x}_{\text{goal}}$	goal position (of an object)
$\boldsymbol{x}^{\text{o}}$	Current object position
$\boldsymbol{x}^{\text{o}}_{\text{f}}$	Final object position
$\boldsymbol{v}^{\text{o}}_{\text{max}}$	Maximum absolute velocity of the object

$\boldsymbol{v}^{\mathrm{o}}$	Object current velocity
$\boldsymbol{k_1}$	Arbitrary constant
$\boldsymbol{x}^{\mathrm{o}}_{\mathrm{des}}$	Desired object position
$\boldsymbol{x}^{\mathrm{o}}_{n}$	Position of next cell location in desired object path
dt	sampling time step
$\boldsymbol{x}^{a}$	Current end-effector position/orientation vector
$\boldsymbol{x}^{a}_{\mathrm{f}}$	Final end-effector position/orientation vector
$\boldsymbol{v}^{a}$	End-effector current velocity
$\boldsymbol{v}^{a}_{\mathrm{max}}$	End-effector maximum velocity
$\boldsymbol{x}^{a}_{\mathrm{des}}$	End-effector desired position/orientation
$\boldsymbol{x}^{a}_{n}$	Next End-effector position/orientation vector in desired path
$\boldsymbol{K}$	Stiffness matrix
$\boldsymbol{T}^{\mathbf{g}}_{\boldsymbol{p}}$	Goal Twist of an arbitrary controller p
$\boldsymbol{\epsilon}$	Machine precision/tolerance (very small value)
$\boldsymbol{\theta}_{\mathbf{init}}$	End-effector initial orientation vector (Euler angles)
$\boldsymbol{p}_{\mathbf{init}}$	End-effector initial position vector
$\boldsymbol{\theta}_{\mathbf{goal}}$	End-effector goal orientation vector (Euler angles)
$\boldsymbol{p}_{\mathbf{goal}}$	End-effector goal position vector
$\boldsymbol{p}^{\mathbf{o}}_{\mathbf{init}}$	Object initial position vector
$\boldsymbol{p}^{\mathbf{o}}_{\mathbf{goal}}$	Object goal position vector