

**OPTIMAL ROBOT TRAJECTORY PLANNING USING
EVOLUTIONARY ALGORITHMS**

BHANU KUMAR GOUDA

Bachelor of Engineering in Electrical and Electronics Engineering

Kakatiya Institute of Electrical and Electronics Engineering

July, 2003

submitted in partial fulfillment of requirements for the degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

at the

CLEVELAND STATE UNIVERSITY

May, 2006

This dissertation has been approved
for the Department of Electrical and Computer Engineering
and the College of Graduate Studies by

Dissertation Committee Chairperson, Dan Simon

Department/Date

Thesis Committee Member, Dr. Majid Rashidi

Department/Date

Thesis Committee Member, Dr. Yongjian Fu

Department/Date

ACKNOWLEDGEMENT

I would like to express my sincere indebtedness and gratitude to my thesis advisor, Dr. Dan Simon, for all his time and effort. His guidance and input truly helped me navigate this endeavor.

I would also thank my committee members, Dr. Majid Rashidi and Dr. Yongjian Fu, for taking time out of their schedules to read my thesis and be on my committee.

I would like to acknowledge Mr. Jatin Bhatt, who was my advisor and mentor while I was working at Rockwell Automation. Throughout this process, there have been many supportive people at Rockwell Automation, too many to thank right now. But suffice it to say, thank you all for giving me a sounding board and encouragement for my ideas.

Finally, I wish to thank my friend Saurabh Jain for his encouragement and intellectual input during the entire course of this thesis.

OPTIMAL ROBOT TRAJECTORY PLANNING

USING EVOLUTIONARY ALGORITHMS

BHANU GOUDA

ABSTRACT

In the last decade, much research has been proposed concerning trajectory generation for manipulators. Also, evolutionary algorithms have been applied in a plethora of fields such as control, robotics, image processing, pattern recognition and speech recognition. In this thesis, we present an optimal trajectory planning approach using evolutionary methods for an industrial manipulator system. Minimum energy consumption is used as a criterion for trajectory generation, and is achieved using genetic algorithms as an optimization tool. We propose the use of cubic spline curves to generate the trajectory between the intermediate points of the path. The problem of kinematics is solved for two-degree-of-freedom linear industrial manipulators. The Newton-Euler technique is used for the formulation of the dynamic equations of the manipulator. The effectiveness of the proposed method is verified through MATLAB simulations.

TABLE OF CONTENTS

	Page
LIST OF TABLES	VII
LIST OF FIGURES	VIII
INTRODUCTION.....	1
1.1 Literature survey	1
1.2 Thesis organization	4
ROBOT MANIPULATORS	6
2. 1 Definition of terms	7
2.2 Coordinate systems and reference frames.....	8
2.3 Coordinate rotation	8
2.4 Manipulator forward and inverse kinematics	12
TRAJECTORY GENERATION.....	21
3.1 Requirements of a trajectory	22
3.1 Joint space trajectory schemes	22
3.2 Cartesian space trajectory schemes.....	28
3.3 Trajectory generation at update period	35
MANIPULATOR DYNAMICS.....	37
4. 1 Dynamics of a rigid body.....	38

4.2 Applications of dynamic analysis	38
4.3 Lagrange-Euler formulation.....	39
4.4 Newton-Euler formulation	40
4.5 Manipulator energy-torque relation	47
EVOLUTIONARY ALGORITHMS	51
5.1 Introduction to genetic algorithms	52
5.2 Basic components of a genetic algorithm	56
5.3 Genetic algorithms approach to the manipulator trajectory problem	63
RESULTS AND CONCLUSIONS	67
6.1 Trajectory analysis	68
6.2 Genetic approach to trajectory generation	76
6.3 Matlab/SimMechanics graphics.....	84
6.4 Conclusions and future work	86
REFERENCES.....	89

LIST OF TABLES

Table	Page
TABLE I: Velocity computed for different intervals in joint space	72
TABLE II: Velocity computed for different intervals in cartesian space.....	75
TABLE III: Energy computed for different intial and final points.....	81

LIST OF FIGURES

Figure		Page
Figure 1:	Rotation matrix interpretation.....	11
Figure 2:	Reachable workspace.....	14
Figure 3:	Right & left arm solutions.....	15
Figure 4:	Manipulator with base offset	15
Figure 5:	Four different solutions.....	16
Figure 6:	Two link manipulator derivation diagram (2 axes).....	18
Figure 7:	Joint trajectories of a two link manipulator	25
Figure 8:	Linear interpolation.....	26
Figure 9:	Linear segment with parabolic blends	27
Figure 10:	Two link manipulator trying to move in path A to B.	32
Figure 11:	High joint velocities near singularity	34
Figure 12:	Start and goal reachable in different solutions.....	35
Figure 13:	Force F acting at the center of mass of body	42
Figure 14:	Moment N acting at the center of mass of body	43
Figure 15:	Two-link manipulator with point masses at distal ends of links.....	45
Figure 16:	Executed operations during a generation.....	55

Figure 17:	Representation of an individual	55
Figure 18:	Crossover for binary representation.....	60
Figure 19:	Binary mutation	62
Figure 20:	Joint1 position (radians) and velocity(radians/sec).....	70
Figure 21:	Joint 2 position(radians) and velocity(radians/sec).....	70
Figure 22:	Joint1 knots at different positions	71
Figure 23:	Joint 1 (radians) trajectory	72
Figure 24:	Joint velocities for different acceleration times	73
Figure 25:	Cartesian trajectory path	73
Figure 26:	Manipulator Cartesian path.....	74
Figure 27:	Vector velocity of the Cartesian move.....	75
Figure 28:	Trajectory of each joint.....	76
Figure 29:	Best individual trajectories	78
Figure 30:	Mean fitness and maximum fitness.....	79
Figure 31:	Joint 2 nearly constant	80
Figure 32:	SimMechanics two link manipulator model diagram	85
Figure 33:	Screen image of the SimMechanics visualization	85

CHAPTER I

INTRODUCTION

“In the study of robotics, the connection between the field of study and ourselves is unusually obvious” [1]. It is for this reason, possibly, that the robotics field interests many of us.

Robotics studies attempt to mimic the behavior of human function by the use of revolute joints, sensors, actuators, controllers and computers. There is much research being pursued in different aspects of the robotics field. The major areas include manipulator design, trajectory planning, artificial intelligence, obstacle avoidance, and computer vision. This thesis work focuses on trajectory planning for manipulators using evolutionary algorithms which connects the first three major areas of robotics fields.

1.1 Literature survey

A significant amount of research has been reported concerning trajectory planning for redundant degree of motion freedom robot manipulators [2]-[3]. Most of them are

based on the calculation of inverse kinematics employing a pseudo-inverse of the Jacobian matrix. Trajectory planning using the minimal-time criterion was proposed under the B-Spline assumption of the Cartesian path [4]. A new scheme based on the variational approach was proposed by Sakamoto [5], in which the trajectory in the joint space is modeled as a B-spline curve, and the performance index is integrated in a straightforward manner through the desired trajectory of the end-effector. Davidor [6] applied genetic algorithms (GAs) to the trajectory generation by searching the inverse kinematics solutions to pre-defined end-effector robot paths.

A new method for time-optimal motion planning based on improved GA was proposed, which incorporates kinematics constraints, dynamics constraints and control constraints of the robotic manipulator [7]. Rana and Zalzala [8] described a method to design a near time-optimal, collision-free motion in the case of multi-arm manipulators. The trajectory planning is carried out in the joint space, and the path is represented by knots connected through cubic splines.

Classical optimization techniques, like dynamic programming, fail to be applicable for applications, in particular for on-line trajectory planning of manipulators, because of their high complexity. Pires and Machado [9] proposed an evolutionary method which optimizes the robot structure and the required manipulating trajectories. They described how an optimal manipulator minimizes both the path trajectory length and the ripple in the time evolution, without any collision with the obstacles in the workspace. An algorithm containing a genetic algorithm and a pattern search is introduced to design the optimal point-to-point trajectory planning for a planar 3-DOF manipulator.

In the area of robotics, one of the major challenges of research is to build autonomous, intelligent robots which have the ability to plan a collision-free path. Roy [11] described a combined fuzzy logic techniques and GA to solve the path planning of a two-link manipulator. In the proposed method, a fuzzy logic controller is used to find obstacle-free directions locally, and GAs are used as optimizers to find optimal locations along the obstacle-free directions. Using *Disjunctive Programming*, a novel algorithm is presented for optimal trajectory planning with obstacles for a 2-DOF manipulator [12].

In recent years the concepts of dynamics and control of manipulators have gained tremendous attention from a number of researchers. The emphasis of research in the area of the manipulators has been on improvement of the generality of mathematical models and the formulation of equations of motions that are amenable to computer solutions and real-time controls for the same. Pasic, Williams and Hui [13] proposed a new solution for the nonlinear differential equations of any order for multibody dynamics and control problems. A planar 2-DOF manipulator is considered for study to validate the algorithm against the popular shooting method.

The large-scale utilization of robot and other electro-mechanical equipment demands a large amount of electrical energy. When the repeated movement of industrial applications is considered, the minimization of energy consumption will be significantly important. This is the main reason that the minimization of electrical energy is examined in this research project. It is observed that the number of robots installed during 2004 in North America was around 15,000 [1]. If each unit consumes around 10 KW-hr, the electrical energy consumed will be around 1.5×10^5 KW-hr; and if we reduce this consumption by 5%, then it would be of significant importance. Commercial electricity

rates range from 5¢ to 18¢ per kilowatt-hour. If we reduce this by 5%, we would be saving around \$1400 per unit for the electrical energy consumed per hour.

1.2 Thesis organization

The current research work focuses on the optimal trajectory planning approach using evolutionary algorithms for an industrial manipulator system. The minimum consumed energy is used as a criterion for trajectory generation, by using a genetic algorithm as an optimization tool.

Chapter II introduces the robotics-related terms followed by the coordinate systems and coordinate rotation. Manipulator forward and inverse kinematics is discussed in the later sections of the chapter.

Chapter III brings in the various schemes for the trajectory generation of a manipulator. Cartesian and joint schemes are discussed under which linear interpolation and spline interpolations are covered. We also describe the geometric problems with Cartesian schemes relating to workspace and singularities.

Chapter IV discusses the governing equations of motion of the multibody systems by Newton-Euler formulation and Lagrange-Euler formulation, and then the torque-energy relation for the manipulator.

Chapter V describes the basic concepts of genetic algorithms, the use of GAs for solving optimization problems and the GA approach to manipulator trajectory generation.

Chapter VI presents the various simulation results for different trajectory schemes and GA simulations for a specific problem of trajectory generation. It then concludes with some conclusions and future work on trajectory generation.

CHAPTER II

ROBOT MANIPULATORS

Robotics research is highly interdisciplinary, requiring the integration of control theory with mechanics, electronics, artificial intelligence, and sensor technology. Robot manipulators are basically multiple degree-of-freedom positioning devices. They are machines that mimic human motion which allow the user to position and orient their “tools”. Because of mechanical considerations, manipulators are generally constructed with joints exhibiting only one degree-of-freedom each. Typically, these are rotational or translational.

In Section 2.1 the robotics-related terms are defined and in section 2.2 and 2.3 the coordinate systems and coordinate rotation are discussed. The manipulator forward and inverse kinematics derivations for the 2-link manipulator are presented in section 2.4. We also discuss the singularities in the workspace in the same section.

2. 1 Definition of terms

Common robotics terms are defined as follows:

Kinematics – The science of motion without regard of the forces giving rise to the motion. We refer to it as a set of mathematical equations that convert positions between two linked geometries.

Forward Kinematics – Computing the Cartesian positions, given the joint positions, is called forward kinematics.

Inverse Kinematics – The solution of joint positions, given Cartesian positions.

Joint Axis – A rotary robotic axis typically having one or more axes of translation or rotation is the joint axis. It can be revolute, prismatic or spherical.

Orientation – Robotic term for directional attitude or rotation about a point in Cartesian space. Orientation is expressed as three ordered rotations around the X, Y, and Z Cartesian axes.

Reference Frame – An Independent Cartesian coordinate system used to define a Cartesian origin and reference orientation with respect to the base coordinate system.

Transformation – General term for conversion equations which map values in one coordinate space to values in another coordinate space.

Translation – Robotic term for a linear movement or offset in Cartesian space. Translation describes the distance between two Cartesian points.

2.2 Coordinate systems and reference frames

We consider that there is a world coordinate system to which everything can be referenced. We describe all positions and orientations with respect to the world coordinate system or with respect to other Cartesian coordinate systems that are defined relative to the world system.

The term reference frame will be used to refer to any independent Cartesian coordinate frame which defines an origin and three primary directions and which can be used to measure Cartesian position. A reference frame must be associated with any coordinate system in which Cartesian position will be measured. For any given Cartesian and non-Cartesian coordinate systems, equations can be developed to convert coordinate positions from one system to the other in either direction, provided their relative position and orientation are known. Transformation equations, which specifically relate Cartesian coordinate systems to non-Cartesian coordinate systems, are often called kinematics. The next section will be focused on rotational transformations.

2.3 Coordinate rotation

Coordinate system rotation can be mathematically represented in the form of a 3×3 matrix. When a position vector $P_s = (X, Y, Z)$ is represented as a column vector, multiplying it by a rotation matrix will transform it into a new position vector $P_t = (A, B, C)$. The rotation transformation equation is:

$$\begin{bmatrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \end{bmatrix} = \begin{bmatrix} X_A & Y_A & Z_A \\ X_B & Y_B & Z_B \\ X_C & Y_C & Z_C \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

This operation can be thought of as rotating a source point $P_s (X, Y, Z)$ by some angle to a new position in space in the target coordinate system called point $P_t (A, B, C)$. It can also be thought of as computing the position of point P_s with respect to a rotated coordinate frame. All angles in all derivations are defined as positive in the counterclockwise direction when viewed along the axis orthogonal to the plane.

The actual transformations for the three primary rotations around the X, Y, and Z axes are:

$$\text{Rotation around the X axis by angle } \gamma : \quad T_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{bmatrix}$$

$$\text{Rotation around the Y axis by angle } \beta : \quad T_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}$$

$$\text{Rotation around the Z axis by angle } \alpha : \quad T_z(\alpha) = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A rotation matrix is orthogonal; i.e., its inverse is equal to its transpose. Compound rotations can also be represented in 3×3 matrix form and can be generated by multiplying the three primary rotation matrices together in sequence as follows:

$$T_{zyx}(\alpha, \beta, \gamma) = T_x(\gamma) T_y(\beta) T_z(\alpha)$$

$$\begin{aligned}
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{bmatrix} \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \\
&\begin{bmatrix} \cos \alpha \cos \beta & \sin \alpha \cos \beta & -\sin \beta \\ \sin \alpha \cos \gamma + \cos \alpha \sin \beta \sin \gamma & \cos \alpha \cos \gamma - \sin \alpha \sin \beta \sin \gamma & -\cos \beta \sin \gamma \\ \sin \alpha \sin \gamma - \cos \alpha \sin \beta \cos \gamma & \cos \alpha \sin \gamma + \sin \alpha \sin \beta \cos \gamma & \cos \beta \cos \gamma \end{bmatrix}
\end{aligned}$$

This is the general expression for rotational transformations expressed as a function of the three primary angles; α , β , γ . Note that the order of the multiplications is important. Matrix multiplication by definition is performed in reverse order of appearance in the $T_x(\gamma) T_y(\beta) T_z(\alpha)$ expression (i.e. rotation around Z is first). When γ , β and α are known, the elements of this matrix become known numbers and can be used to transform positions.

If there are rotations around Z, and then around Y, then the general expression above reduces to [1]:

$$\begin{aligned}
T_{zy}(\alpha, \beta) &= T_y(\beta) T_z(\alpha) = \\
\begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} &= \begin{bmatrix} \cos \alpha \cos \beta & -\sin \alpha \cos \beta & \sin \beta \\ \sin \alpha & \cos \alpha & 0 \\ -\cos \alpha \sin \beta & \sin \alpha \sin \beta & \cos \beta \end{bmatrix}
\end{aligned}$$

Similarly other combinations of rotations around the primary axes can be done.

Interpretation of the rotation matrix

The rotation matrix for rotation around the Z axis is:

$$T_z(\alpha) = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The columns of the rotation matrix may be thought of as the projections of three vectors. Each axis of the rotated coordinate system X' , Y' , Z' is projected onto the three axes of the un-rotated coordinate system X , Y , Z as shown in Figure 1.

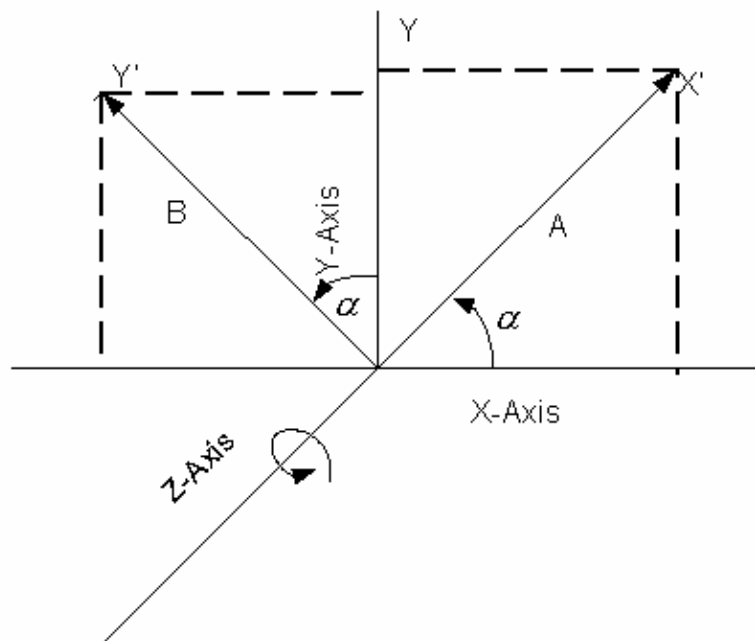


Figure 1: Rotation matrix interpretation

The first column of the matrix can be interpreted as the projection of vector A (i.e., a point on the X' axis), onto the X , Y , Z axes. $\cos \alpha$ is the projection of vector A onto the X axis, $\sin \alpha$ is the projection of vector A onto the Y axis. There is no

projection of vector A onto Z, since the Z axis is out of the plane. The second column is the projection of vector B onto the un-rotated X, Y, Z axes. $\sin \alpha$ is the projection of vector B onto the X axis, $\cos \alpha$ is the projection of vector B onto the Y axis and 0 is the projection of vector B onto the Z axis. Again, there is no projection of vector B onto Z since Z is out of the plane. The third column is the projection of a vector C on the Z axis onto the X, Y, Z axes. Vector C is on the Z axis and therefore is not shown in Figure 1. Its projection onto both the X and Y axes is 0. Its projection onto Z is 1.

2.4 Manipulator forward and inverse kinematics

In the study of robotics, we are constantly concerned with the location of objects in three-dimensional space. At a crude but important level, these objects are described by just two attributes: position and orientation. Kinematics deals with the study of the geometry of motion of a robot arm with respect to a fixed coordinate system without regard to the forces or moments which cause it. Within kinematics, we study the position, velocity, acceleration and all higher order derivatives of the position variables.

In the study of manipulator trajectory design, forward and inverse kinematics are the basic geometrical problems. Forward kinematics is the problem of calculating the position and orientation of the tool center point of the manipulator. In particular, given a set of joint angles of a coordinate system, the forward kinematics problem computes the Cartesian position and orientation of the tool with respect to the base.

Inverse kinematics is a quite complicated geometrical problem that is worked out daily in our body and other biological systems. In the case of a manipulator, we have to develop algorithms which calculate the joint angles that could be used to attain the target position and orientation. Solving the inverse kinematics becomes complex and nonlinear as the number of degrees of freedom of the manipulator increases.

2.4.1 Workspace of manipulator

Workspace is the space that the tool center point of the robot can reach. The manipulator workspace can be dexterous or reachable. The dexterous workspace is defined as the space in which the manipulator end-effector can reach with all orientations. The reachable workspace is the volume of space that the robot can reach in at least one orientation [1].

Let us consider the workspace of the two-link manipulator as shown in the Figure 2. If $l_1=l_2$, then the reachable workspace consists of a circle of radius $2l_1$. If $l_1 \neq l_2$, the reachable workspace would be a circle of outer radius $l_1 + l_2$ and inner radius $|l_1 - l_2|$. We assume the ideal case where the manipulator has both of the joint limits between zero degrees and 360 degrees. If the manipulator has base and end-effector offsets, then the reachable workspace would be different.

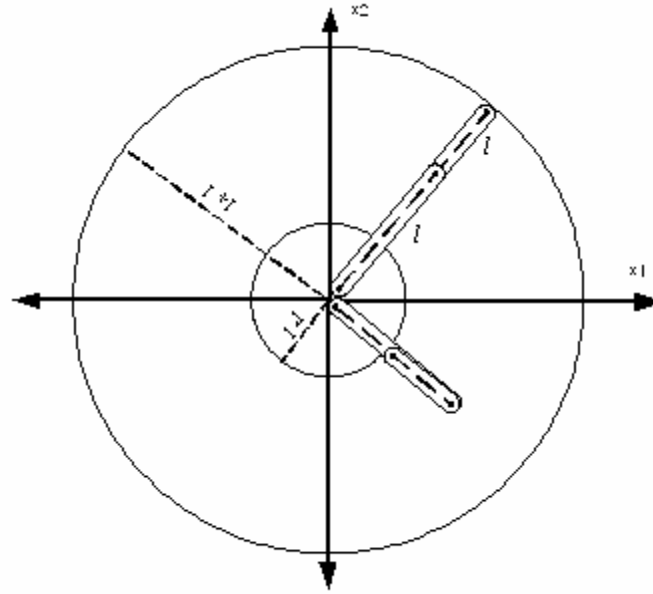


Figure 2: Reachable workspace

2.4.2 Different solutions for single Cartesian point

Another problem we usually encounter in solving kinematics equations is that of multiple solutions. Even this depends on the degrees of freedom that the manipulator has. In general, a two link manipulator in the 2-D plane will have two kinematics solutions at each position (point (x_1, x_2) in the following Figure 3). One solution will satisfy the equations for a right-armed robot, the other for a left-armed robot. We interpret the right- and left-arm solution based on the value of J_2 . If J_2 is negative, then it is a right-arm solution, and if the value of J_2 is positive, it is left-arm solution.

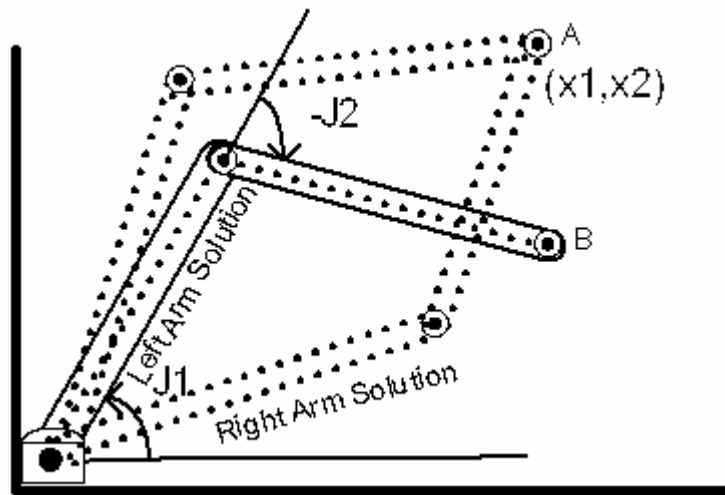


Figure 3: Right & left arm solutions

If we consider the 2-link manipulator shown in Figure 4 with three degrees of freedom, where the base rotates around Z axis, we will have four different solutions for a single Cartesian point as shown in Figure 5. Figure 4 is a drawing of an Adept6 manipulator which is popularly used in many applications [1].

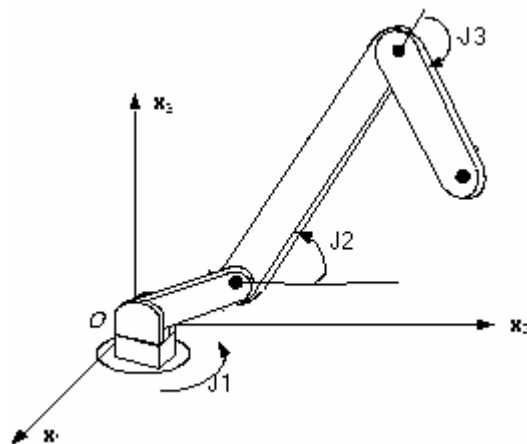


Figure 4: Manipulator with base offset

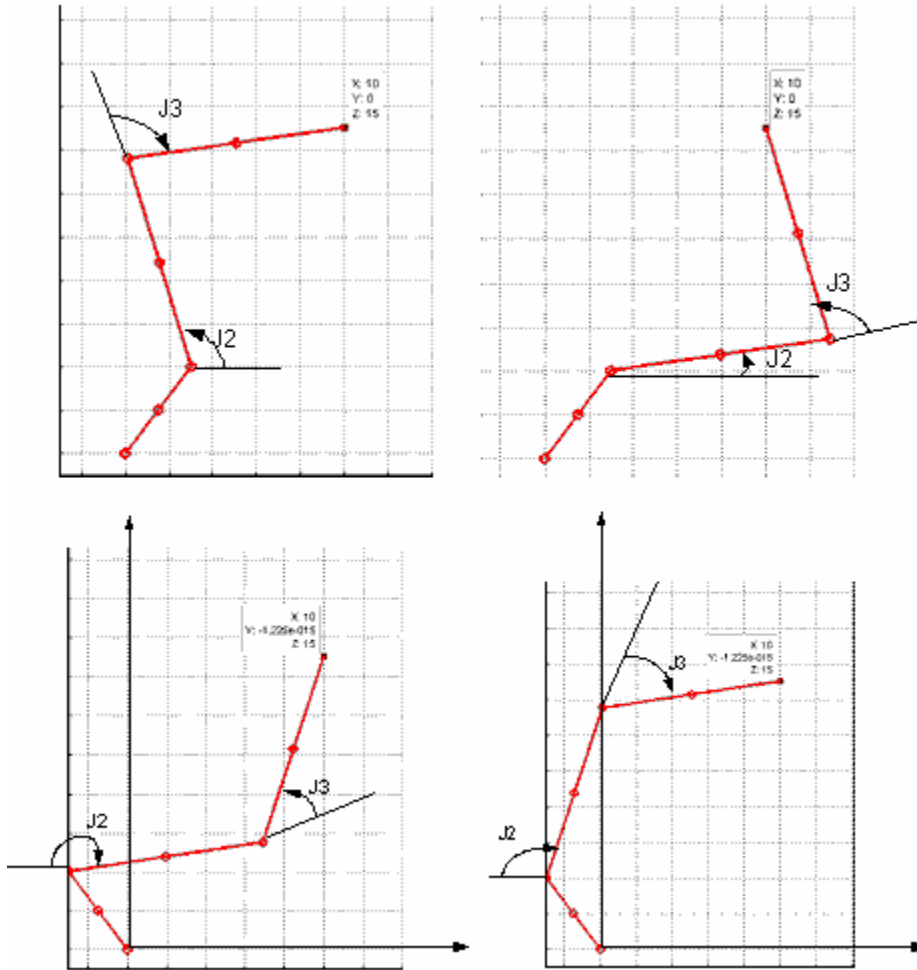


Figure 5: Four different solutions

The criteria upon which solution to choose from the multiple solutions would be the closest solution. For example, robots having three larger links, followed by three smaller, orienting links near the tool center point of the manipulator. In this case, weights should be applied for the calculation of which solution is closer so that the selection favors moving smaller joints rather than moving larger joints [1].

2. 4.3 Kinematics derivations

There are two different methods of solving the kinematics equations: algebraic and geometric. The algebraic approach to solving kinematics equations is basically manipulating the given equations into a form for which a solution is known. It is found that for most common geometries, several forms of transcendental equations arise. In the geometric approach, we decompose the geometry of each arm into several plane-geometry problems. With the geometric approach, the forward and inverse kinematics solutions for a two link manipulator are derived and are based on the derivation diagram shown in Figure 6.

Forward kinematics transformations

The algebraic relationship between the coordinates (α_1, α_2) of a point P_s specified in the robot system and the coordinates (x_1, x_2) of the same point P_t specified in the Cartesian reference system is defined by the following equation [7].

$$P_t = f_{\text{Indep.} \rightarrow \text{Cart.}}(P_s)$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} l_1 \cos \alpha_1 + l_2 \cos(\alpha_1 + \alpha_2) \\ l_1 \sin \alpha_1 + l_2 \sin(\alpha_1 + \alpha_2) \end{bmatrix}$$

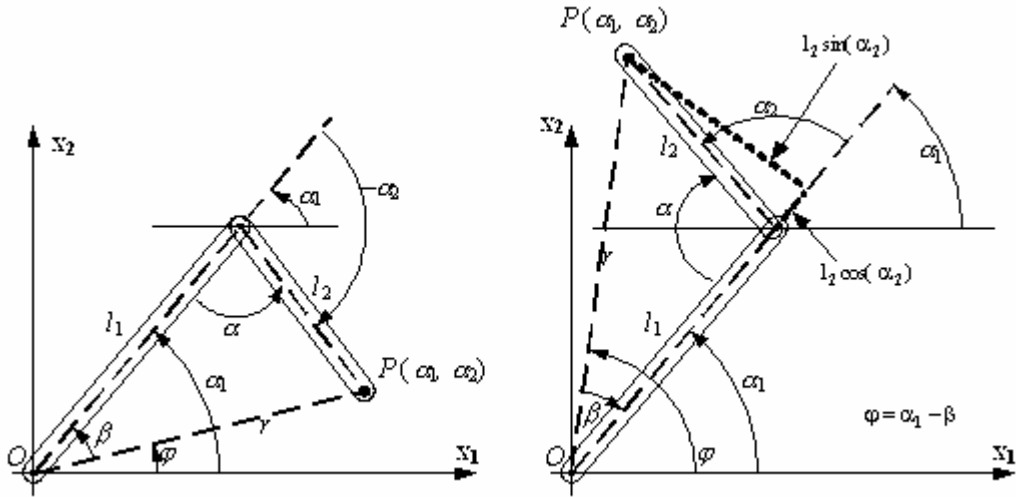


Figure 6: Two link manipulator derivation diagram (2 axes)

Inverse kinematics transformations

Since the kinematics system can either represent a left-arm ($\alpha_2 < 0$) or a right-arm ($0 \leq \alpha_2$) system, the result of this transformation is not uniquely defined. However, by establishing one configuration during initialization, the result is uniquely defined from then on.

With the polar coordinates of a point P_s

$$\varphi = \text{atan2}(x_2, x_1)$$

and the cosine rule, the coordinates (α_1, α_2) can be derived. First, the cosine rule

$$\begin{aligned} r^2 &= l_1^2 + l_2^2 - 2l_1l_2 \cos \gamma = l_1^2 + l_2^2 - 2l_1l_2 \cos(\pi - \alpha_2) \\ &= l_1^2 + l_2^2 + 2l_1l_2 \cos \alpha_2 \end{aligned}$$

with the definition

$$a = \cos \alpha_2 = \frac{r^2 - l_1^2 - l_2^2}{2l_1 l_2}$$

and

$$b = \sin \alpha_2 = \pm \sqrt{1 - \cos^2 \alpha_2} = \pm \sqrt{1 - a^2}$$

where the positive sign defines a right-arm and the negative sign defines a left-arm configuration, provides the angle

$$\alpha_2 = \text{atan2}(b, a)$$

Second, the geometry

$$\tan \beta = \frac{l_2 \sin \alpha_2}{l_1 + l_2 \cos \alpha_2} = \frac{l_2 \sin \alpha_2}{l_1 + l_2 \cos \alpha_2} = \frac{l_2 b}{l_1 + l_2 a}$$

provides the angle

$$\alpha_1 = \varphi - \beta = \text{atan2}(x_2, x_1) - \text{atan2}(l_2 b, l_1 + l_2 a)$$

The algebraic relationship between the coordinates (x_1, x_2) of a point P_s specified in the Cartesian reference system, and the coordinates (α_1, α_2) of the same point P_t specified in the robot system, is defined by the following equations.

$$P_t = f_{\text{Cart.} \rightarrow \text{Indep.}}(P_s)$$

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} \text{atan2}(x_2, x_1) - \text{atan2}(l_2 b, l_1 + l_2 a) \\ \text{atan2}(b, a) \end{bmatrix}$$

2.4.4 Singularities in manipulator workspace

At a singular configuration, the end-effector locally loses the ability to move along or rotate about some direction in Cartesian space [14]. All manipulators have singularity conditions at the boundary or very near to the boundary of their workspace. A singularity condition arises when the robot is fully stretched or folded back on itself.

At singularities, or near singularities, we have high joint rates which are discussed in the simulation results chapter. We will also discuss the various geometric problems involved in Cartesian trajectory paths in the next chapter.

CHAPTER III

TRAJECTORY GENERATION

Robots are built to accomplish complex and difficult tasks that require highly non-linear motions. Specifying the desired motion to achieve a specified goal is often a difficult task. In this chapter, we present different methods of computing the trajectory as a function of position, velocity and acceleration for each joint of the manipulator. The trajectory we are interested in tracking is expressed in the end-effector coordinates. The user should be able to specify nothing more than the desired goal position and orientation of the end-effector trajectory time, and leave it to the system to decide on the exact shape of the path to get there, the velocity profile, and other details.

In section 3.1 we discuss the basic requirements of trajectory generation. Then we talk about the joint space schemes under which we describe the cubic spline polynomial approach and then linear joint schemes. We present the Cartesian schemes where we discuss how to generate the trapezoidal Cartesian straight line point-to-point trajectory in the end-effector in section 3.2. We also describe the geometric problems with Cartesian paths in this section.

3.1 Requirements of a trajectory

The basic requirements for the trajectory planning are:

1. The trajectory should be specified relative to the station frame. We should allow the generalization of moving station frames without significant problems. Moving conveyor belts can be considered as an example.
2. The trajectory should be smooth, i.e., the position and its first derivative should be smooth. This avoids wear on joint motors and also reduces impulsive forces applied to the payload. Various algorithms are available for computing the smooth functions [15].
3. A trajectory should satisfy the temporal requirements of the task. Many applications require fairly specific rates of motion or a specific time for completion. For example, a conveyor belt may be stationary only for a short time to allow a weld to be made, or in spray painting applications the movement of the tool piece is more important than its placement.

3.1 Joint space trajectory schemes

In this section, we discuss trajectory generation methods in which the paths are described in terms of functions of joint angles. The joint angles are generated using the inverse kinematics of the manipulator from the user-defined Cartesian coordinates. Joint space schemes are usually easy to compute and there is no problem with singularities.

3.1.1 Cubic spline approach

A common way of causing a manipulator to move from point to point in a smooth controlled fashion is to cause a joint to move as specified by a smooth function of time. Commonly, all joints start and end their motion at the same time, so that the manipulator appears to be coordinated. Exactly how to compute these motion functions is the problem of **trajectory generation**. We will consider the trajectory of a manipulator as motions of the tool frame with respect to the stationary frame, so that we can separate the motion descriptions from any particular robot or end-effector. This results in the flexibility of allowing the path description to be used for different manipulators, or the same manipulator with a different tool size.

The fundamental problem is to move the tool frame from its current Cartesian position to the goal position, where the motion involves both a change in position and orientation. Usually it would be essential to specify the motion in much more detail than by simply specifying the desired goal position. One way is to include a sequence of desired via points called **knots in the trajectory path**.

Along with spatial constraints on the motion, the user should specify the time elapsed between via points in the description of the path. In this work we consider the trajectory knots to be spaced at regular intervals of time.

As we discussed before the basic requirement for trajectory generation is that the path should be smooth. For our case, we will describe a smooth function that is continuous and has a continuous first derivative. If the generated trajectory is rough and

jerky, then it causes wear on the mechanism and cause vibrations by exciting resonances in the manipulator.

Cubic spline functions are the most popular spline functions for many reasons. They are smooth functions, and, when used for interpolation, they do not have the oscillatory behavior that is characteristic of high-degree polynomial interpolation.

The functions that have been most frequently used for the mathematical development of splines are the simple univariate or bivariate polynomials. Cubic Splines have a special place among these functions. A cubic spline function is made by joining various univariate or bivariate cubic polynomials, i.e., by defining the cubic spline function of interest piece-by-piece, where each piece is given by a particular cubic polynomial [16].

The trajectories for the two-link manipulator are as shown in Figure 7. The trajectory time is defined by the user, and the knots are at regular intervals of time.

Consider the trajectory time,

$$T_i \leq t_0 < t_1 < \dots < t_n \leq T_f$$

For a single joint we have

$$\theta(t_i) = \theta_i \quad \text{where } i = 0, 1 \dots n \quad \text{and} \quad (3.1)$$

$$\theta_i(t) = a_i + b_i t + c_i t^2 + d_i t^3 \quad t_{i-1} \leq t \leq t_i \quad i = 1, 2, \dots, n \quad (3.2)$$

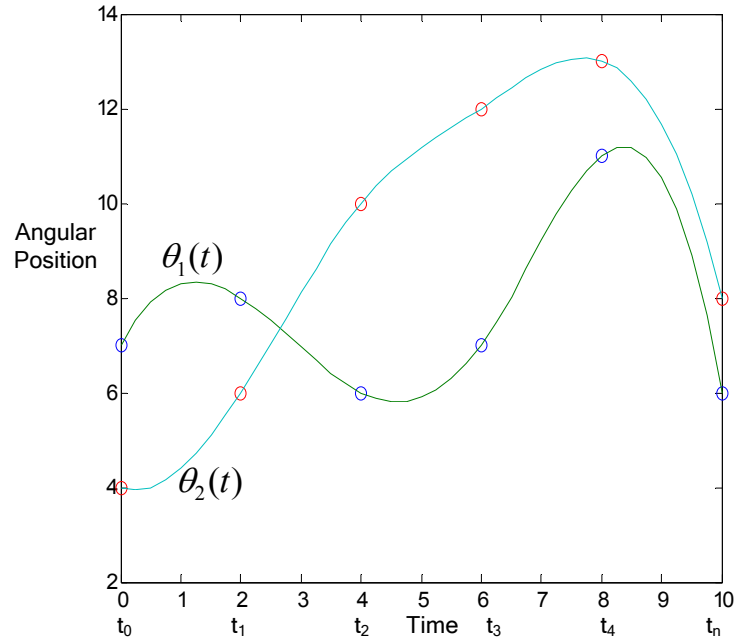


Figure 7: Joint trajectories of a two link manipulator

where $(n - 1)$ is the number of knots between the initial and final knot positions. There are n cubic polynomials to be found, each having four unknown coefficients. Thus, the set of equations to be solved involves $4n$ unknown coefficients. The i th spline will be evaluated over an interval starting at $t = t_i$ and ending at $t = t_{i+1}$, where $i = 0, 1, 2, \dots, (n - 1)$. To obtain the coefficients we need to have $4n$ constraints. The constraints are Equation (3.3) and the continuity restrictions

$$\theta^j(t_i^+) = \theta^j(t_i^-) \quad i = 1 \dots (n - 1) \quad j = 0, 1, 2 \quad (3.3)$$

Together this gives $n + 1 + 3(n - 1) = 4n - 2$ constraints, as compared with $4n$ unknowns.

For our interpolation problem, we have two more degrees of freedom in choosing the coefficients of Equation 3.2. The manipulator is considered to be at zero velocity at the start and end positions.

$$\text{So, } \dot{\theta}_1(t_0) = 0 \quad \text{and} \quad \dot{\theta}_n(t_n) = 0$$

Solving the $4n$ constraint linear equations, we get the cubic spline coefficients which result in describing the trajectory of the joint passing through the specified knots.

3.1.2 Linear function with parabolic blends

A simpler interpolation scheme than the polynomial is a linear interpolation. That is, we interpolate linearly between the initial and final joint position as shown in Figure 8. Although the motion of each joint is linear, the end-effector in general does not move in a straight line in space. The problem is that at via points, the velocity and acceleration will be discontinuous. A good solution would be to add a “parabolic blend” section at the via point to interface the two interpolating curves. During the blend portion of the curve, we choose the blend so that constant acceleration is used to change the velocity smoothly. Figure 9 shows a simple path constructed in this way.

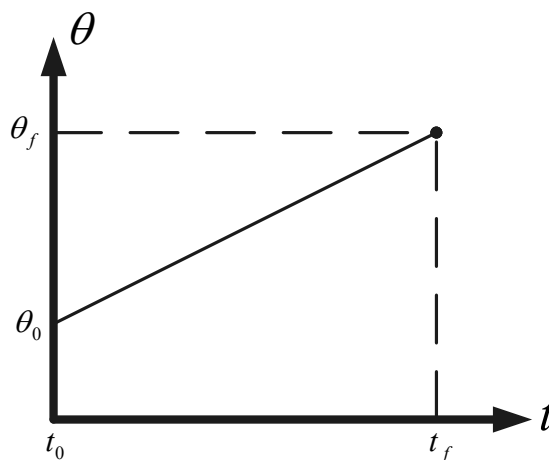


Figure 8: Linear interpolation

Depending on the value of acceleration chosen, and on the change in velocity required between adjacent linear sections, the blend region will extend further or less into the linear region [1].

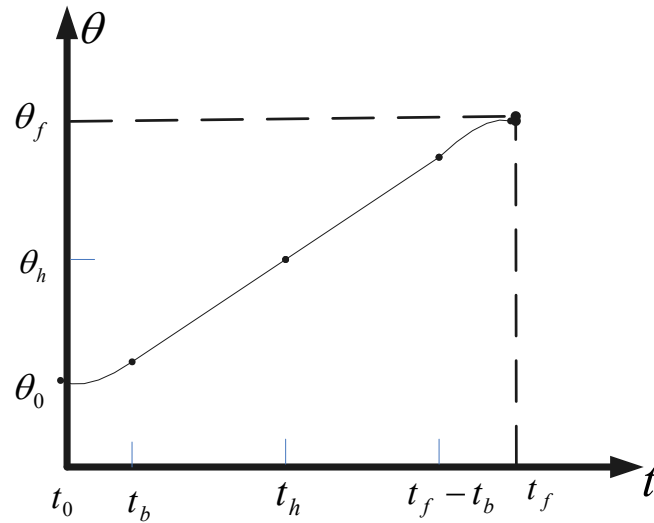


Figure 9: Linear segment with parabolic blends

There can be another case of linear interpolation where we interpolate linearly in joint space. We choose a trapezoidal velocity profile along the complete path to produce a joint trajectory.

For this trajectory we define the trajectory time for the whole coordinate move. Even though the interpolation is linear in joint space, the individual joint trajectories and also the end-effector Cartesian path will not be a straight line. For the reason the vector length of each joint move would be different over the same time.

3.2 Cartesian space trajectory schemes

We discussed in the previous section the paths computed in the joint space, where in the start, via and end points are reached even they are specified in a Cartesian frame. But the path followed by the manipulator was not a straight line connecting the start and end points. Rather it would be some complicated shape that depends on the trajectory approach.

Cartesian trajectories are the best representation of the actual motion of the end-effector of the robot. However, Cartesian motion of the robot does not map trivially to a trajectory in joint space. The trajectory resulting from a Cartesian path is generally more complex in joint space than a direct interpolation and can lead to high actuation requirements on individual joints. On the positive side, for Cartesian trajectories the motion of the end-effector is usually smooth and natural. Consequently there are reduced inertia and gyroscopic disturbances on the manipulator because of the load carried by the end-effector.

In Cartesian paths joint motion is obtained via a repeated application of the inverse kinematics at every point along the trajectory. Since trajectories are not generated in joint space, care must be taken that the path lies in the reachable work space and does not pass through singularities.

3.2.1 Path planning algorithm

Various schemes for generating Cartesian paths are proposed in the literature [17, 18]. The normal path planning algorithm plans the profile with specified initial velocity, current position, and distance to travel. In case of a time-based planner, the time for the move (T_e) is specified instead of the target velocity.

The initial conditions for the interpolator as used by the algorithm are denoted as follows:

T_e = Time for the move

D_{tg} = Distance to travel

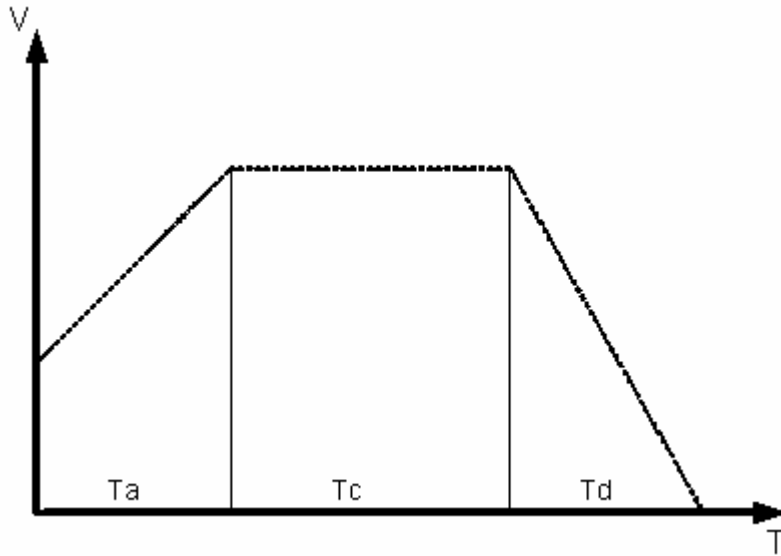
V_i = Initial velocity

Acc = Specified acceleration

Dec = Specified deceleration

V_{max} = maximum target velocity for the interpolator.

The following figures show the two profiles types based on the initial conditions, V_{max} , and total time, T_e . The algorithm determines which profile type is needed.

Profile Type – A

$$cart1 = \{X1, Y1\}; cart2 = \{X2, Y2\};$$

Distance traveled for Type-A =

$$Dtg = norm(cart1 - cart2) = \text{Accel Distance} + \text{Coast Distance} + \text{Decel Distance}$$

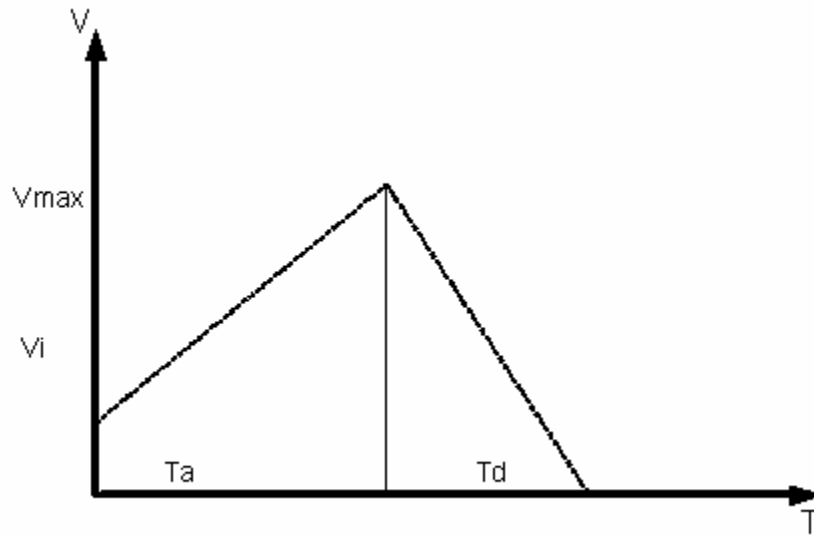
$$\text{Accel Distance} = DTa = \frac{V_{\max}^2 - V_i^2}{2 \text{Acc}}$$

$$\text{Decel Distance} = DTd = \frac{V_{\max}^2}{2 \text{Dec}}$$

$$\text{Coast Distance} = DTc = Dtg - DTa - DTd$$

$$Ta = \frac{V_{\max} - V_i}{\text{Acc}}; Td = \frac{V_{\max}}{\text{Dec}}; Tc = \frac{DTc}{V_{\max}}$$

$$Te = \text{Total time} = Ta + Tc + Td;$$

Profile Type - B

Distance traveled for Type-B =

$$DTg = \text{norm}(\text{cart1} - \text{cart2}) = \text{Accel Distance} + \text{Decel Distance}$$

$$\text{Accel Distance} = DTa = \frac{V_{\max}^2 - V_i^2}{2 \text{Acc}}$$

$$\text{Decel Distance} = DTd = \frac{V_{\max}^2}{2 \text{Dec}} \quad \text{Since } DTc \leq 0 \quad Tc = 0;$$

$$Ta = \frac{V_{\max} - V_i}{\text{Acc}} \quad ; \quad Td = \frac{V_{\max}}{\text{Dec}} \quad ; \quad Te = \text{Total time} = Ta + Td;$$

$$V_{\max} = \sqrt{\frac{2 \text{DTG}}{\frac{1}{\text{Acc}} + \frac{1}{\text{Dec}}}}$$

At runtime the path generator routine constructs the trajectory at the path-update rate, and the inverse kinematics routine transforms the Cartesian information into joint angles which are fed to the manipulator's control system.

3.2.2 Problems with Cartesian paths

Points not reachable

In Cartesian paths, even though the initial point and the final point are in the reachable workspace, it is possible that not all points which are on the straight line between these two points are in the workspace. For instance, consider the two link manipulator as shown in Figure 10. In this case, link1 is greater than link2, so the workspace contains an inner radius in the middle whose radius is the difference between the link lengths. If we draw a straight line starting from the initial point A to a goal point B and attempt to make a Cartesian move, the intermediate points will not be reachable. In that case we need to employ joint space schemes, which are an advantage over Cartesian schemes [1].

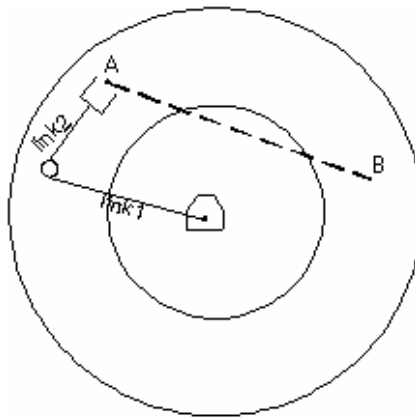


Figure 10: Two link manipulator trying to move in path A to B.

Joint velocities near singularity

We already discussed in section 2.4.4 the singularities in the workspace. It is impossible to limit the joint velocities that yield the desired Cartesian velocity for the

end-effector. If, for example, a manipulator is following a Cartesian straight line path and approaches a singular configuration of the mechanism, one or more joint velocities will increase towards infinity.

As an example, Figure 11 shows a two-link manipulator with equal link lengths moving along a path from point A to point B. The desired path is to move the tool tip along this straight line maintaining a constant linear velocity. All points along the path would be reachable, but as the robot gets near to the singularity, which is the origin in this case, the velocity of the joints becomes very high. This problem is also observed when the manipulator is getting near to the fully stretched singularity condition.

There might be cases where one is required to follow a Cartesian path which approaches the singularity condition. One solution would be to program the system in such a way that the move is completed in three separate moves.

The first programmed move will be a Cartesian path from the initial point to a point very close to the origin. Then a very small linear joint move is programmed, followed by the Cartesian path to the end point.

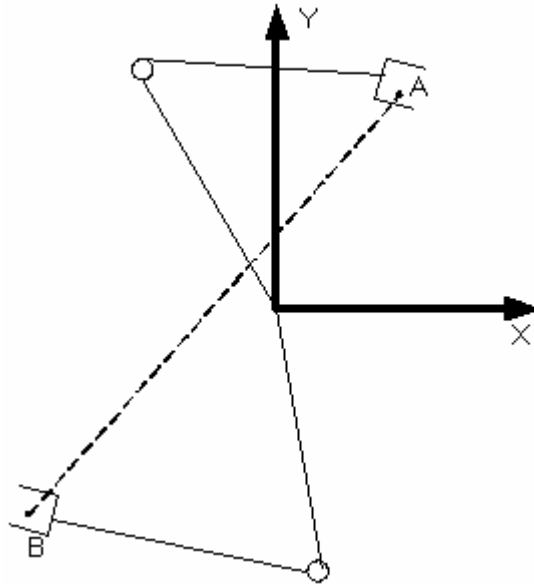


Figure 11: High joint velocities near singularity

Lefty- Righty solutions with cartesian paths

We discussed in section 2.4.2 that, for a single Cartesian point there, are multiple ways of approaching the point. In case of the two-link manipulator, it would be a right arm solution and a left arm solution.

For example, if we want to program a Cartesian path from point A to point B, as shown in Figure 12, we can see that by the time it reaches the final point, the approach solution is changed from a right arm configuration to a left arm configuration.

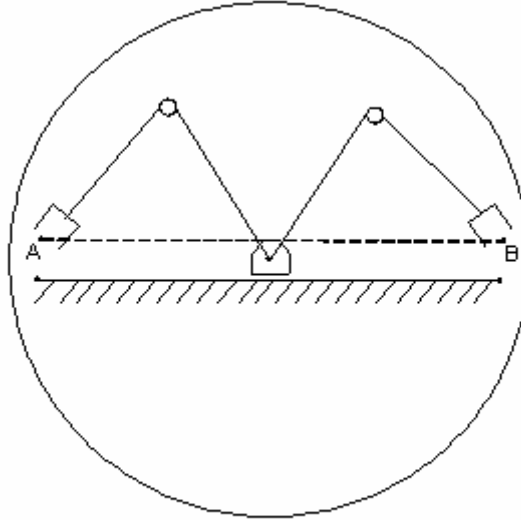


Figure 12: Start and goal reachable in different solutions

Because of the difficulties with Cartesian paths, joint space paths should be used as the default, and Cartesian space paths should be used only when actually needed by the application.

3.3 Trajectory generation at update period

For all the above trajectory schemes, the final trajectory path is a set of data for each segment of the trajectory. At run time, the interpolator routine generates the trajectory position, velocity and acceleration, and feeds the information to the manipulators control system at the path update period.

In the case of cubic splines, the path generator simply computes as t is advanced. When the end of one segment is reached, a new set of cubic coefficients is recalled, t is

set back to zero, and the generation continues. In the case of linear splines with parabolic blends, the value of time, t , is checked on each update to determine whether we are currently in the linear or blend portion of the segment.

Because a continuous correspondence is made between a path shape described in Cartesian space and joint positions, Cartesian paths are prone to various problems relating to workspace and singularities.

In this work, a cubic spline approach for optimal trajectory generation is employed and compared against linear interpolation. The amount of acceleration that the manipulator is capable of at any instant of time is a function of the dynamics of the arm and the actuator limits. In the next chapter the dynamics of the manipulators are discussed in detail.

CHAPTER IV

MANIPULATOR DYNAMICS

In the last chapter we concentrated on kinematics and its trajectory generation only. We studied the coordinate systems and the manipulator kinematics, but we did not consider the forces that cause the motion. In this chapter we will consider the motion equations for a manipulator and how it results in the torques applied to the joint motors or actuators.

Dynamics is a wide field of study devoted to studying the forces required to cause motion [1]. In order to accelerate a manipulator from rest, glide at a constant end-effector velocity, and finally decelerate to stop, a complex set of torque functions must be applied by the joint actuators.

In Section 4.1 the various dynamic analysis techniques are defined followed by applications of the dynamic analysis in Section 4.2. The Lagrange-Euler formulation is discussed in Section 4.3 and the Newton-Euler formulation in Section 4.4. The Newton-Euler formulation covers the dynamics of rigid links and the computation of torque. The chapter ends with a discussion of the manipulator torque-energy relationship in Section 4.5.

4.1 Dynamics of a rigid body

Several methods of dynamic analysis have been developed and employed to overcome the computational difficulty in dynamic multibody systems. All these require reforming the kinematics, dynamics and inverse dynamic procedures and the design organization of the computers. The real dynamic model of a robot arm is obtained from known physical laws, such as the laws of Newtonian mechanics and Lagrangian mechanics, which lead to the development of the dynamic equations of motion for the various joints of the manipulator. Generally, the governing equations of motion of the multibody systems can be derived by a number of methods including Newton-Euler equations, Lagrange equations, d'Alembert's principle and Kane's method. Recursive formulations that use relative coordinates offer the best method of calculation.

4.2 Applications of dynamic analysis

The dynamic analysis finds essential use in trajectory planning and control of the robotic manipulators, usually when paths are planned at default or at maximum acceleration at the blend points. Nevertheless the amount of acceleration that the manipulator is capable of at any instant is a function of the dynamics of the arm and the actuator limits. Most actuators are not characterized by a fixed maximum torque or acceleration, but rather by a torque-speed curve. When a trajectory is planned assuming that there is a maximum acceleration at each joint, a tremendous simplification is made and full use of the speed capabilities of the manipulators cannot be made. The solution

for finding the minimum time trajectory and minimum energy makes use of the manipulator dynamics.

4.3 Lagrange-Euler formulation

The derivation of the dynamic model of a manipulator based on the LE formulation is simple and systematic. Presuming a rigid body motion, the resulting motion equations, excluding the dynamics of electronic control devices, backlash, and gear friction, are a set of second-order nonlinear differential equations. Using the 4×4 homogenous transformation matrix representation of the kinematics chain and the Lagrangian formulation, it is found that the dynamic motion equations for a robot arm are highly nonlinear and consist of inertia loading, coupling reaction forces between joints (Coriolis and centrifugal), and gravity loading effects.

The torques and forces depend on the manipulator's physical parameters, instantaneous joint position, velocity and acceleration, and also the payload it is carrying. The LE equations of motion provide explicit state equations for robot dynamics and can be utilized to analyze and design advanced joint-variable space control strategies. They are used to solve for the forward dynamics problem. That is, given the desired torques/forces, the dynamic equations are used to solve for the joint accelerations, which are then integrated to solve for the generalized coordinates and their velocities. They can also be used to solve for the inverse dynamics problem. That is, given the desired generalized coordinates and their first two time derivatives, the generalized

forces/torques are computed. The LE equations are very complex to use for real-time control purposes unless they are simplified.

4.4 Newton-Euler formulation

Generalized forces/torques are developed based on the N-E equations of motion as an alternative for deriving more efficient equations of motion. The derivation involves vector cross-product terms and the resulting dynamic equations are a set of forward and backward recursive equations. This set of recursive equations can be applied to the robot links sequentially. The forward recursion propagates kinematics information (such as linear velocities, angular velocities, angular accelerations, and linear accelerations at the center of mass of each link) from the inertial coordinate frame to the hand coordinate frame. The backward recursion propagates the forces and moments exerted on each link from the end-effector of the manipulator to the base reference frame. The most significant result of this formulation is that the computation time of the generalized forces/torques is linearly proportional to the number of joints of the robot arm and is independent of the robot arm configuration. With this algorithm, simple real-time control of a robot arm in the joint-variable space can be implemented.

4.4.1 Notation

We use the following conventions for the variables [1]:

1. Uppercase variables represent vectors or matrices and the lower case variables are scalars.
2. Subscripts and superscripts before the variable recognize which coordinate system a quantity is in. For example, ${}^A B$ represents a position vector written in coordinate system $\{A\}$, and ${}^A_B R$ is a rotation matrix (3×3) that specifies the relationship between coordinate systems $\{A\}$ and $\{B\}$.
3. Trailing subscripts indicate a vector component (e.g., x, y, z) or may be used as a description, as in P_{bolt} , the position of a bolt.

4.4.2 Manipulator dynamics of rigid links

Comparative studies have shown the computation efficiency of the Newton-Euler approach to exceed the Lagrangian method [19].

We consider each link of a manipulator as a rigid body, and its mass distribution is described based on the center of mass location and the inertia tensor of the link. We should accelerate and decelerate the joints so as to cause motion in the links; the forces required for such motion are a function of the desired acceleration and the mass distribution of the links. Newton's equation and Euler's equations describe how the forces, inertias, and accelerations relate in the manipulator

Figure 13 shows a body whose center of mass is accelerating with acceleration \dot{v}_c . The force F acting at the center of mass and causing this acceleration is given by Newton's equation $F = m\dot{v}_c$, where m is the total mass of the body [1].

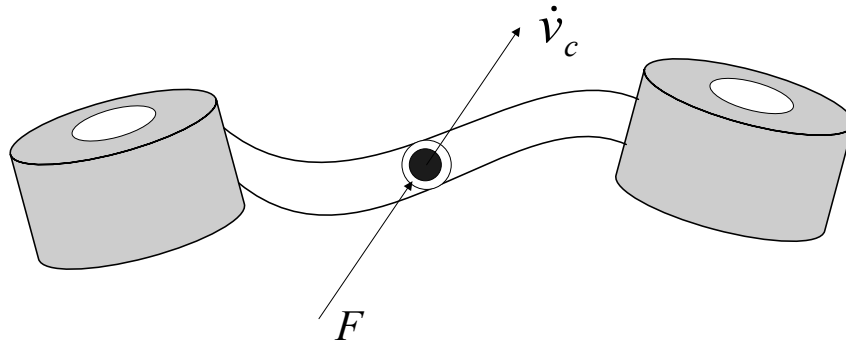


Figure 13: Force F acting at the center of mass of body

Figure 14 shows a rigid body rotating with angular velocity ω and with angular acceleration $\dot{\omega}$. The moment N , which is acting on the body at the center of mass to cause this motion, is given by Euler's equation $N = {}^c I \dot{\omega} + \omega \times {}^c I \omega$, where ${}^c I$ is the inertia of the body. The Newton-Euler technique is used for the formulation of equations of motion for a manipulator, which is basically a force-momentum balance technique. Luh's strategy [20] has been followed which refers all velocities, accelerations, inertial matrices, and forces/moments to their own link coordinate systems. This enables real-time control for the robot arm in joint variable space.

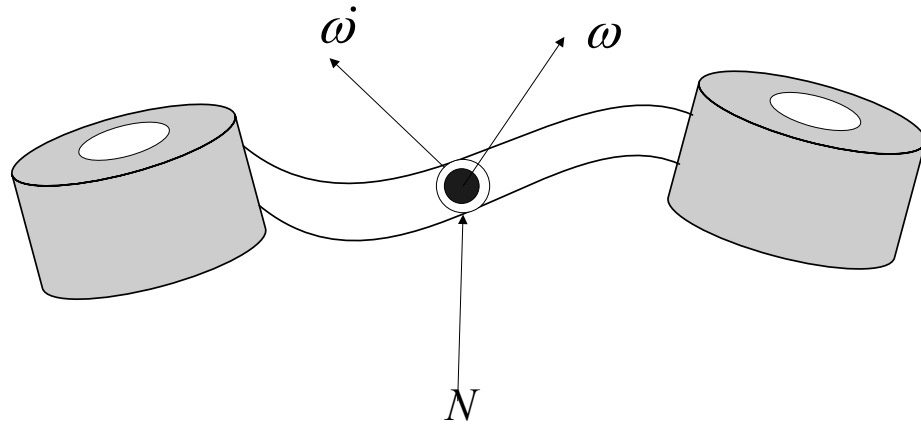


Figure 14: Moment N acting at the center of mass of body

The complete algorithm is in two parts:

- a) First, link velocities and accelerations are iteratively computed from link 1 to link n , and the Newton-Euler equations are applied to each link to compute inertia forces and moments.
- b) Second, forces and torques of interaction and joint actuator torques are computed recursively from link n back to link 1.

Outward iterations: $i: 1 \Rightarrow n$

$$\left. \begin{aligned}
{}^{i+1}\omega_{i+1} &= {}^{i+1}R^i \omega_i + \dot{\theta}_{i+1} \hat{Z}_{i+1} \\
{}^{i+1}\dot{\omega}_{i+1} &= {}^{i+1}R^i \dot{\omega}_i + {}^{i+1}R^i \omega_i \times \dot{\theta}_{i+1} \hat{Z}_{i+1} + \ddot{\theta}_{i+1} \hat{Z}_{i+1} \\
{}^{i+1}\dot{v}_{i+1} &= {}^{i+1}R^i (\dot{\omega}_i \times {}^iP_{i+1} + \omega_i \times (\omega_i \times {}^iP_{i+1}) + \dot{v}_i) \\
{}^{i+1}\dot{v}_{C_{i+1}} &= {}^{i+1}\dot{\omega}_{i+1} \times {}^{i+1}P_{C_{i+1}} + \omega_{i+1} \times (\omega_{i+1} \times {}^{i+1}P_{C_{i+1}}) + \dot{v}_{i+1} \\
{}^{i+1}F_{i+1} &= m_{i+1} {}^{i+1}\dot{v}_{C_{i+1}} \\
{}^{i+1}N_{i+1} &= {}^{Ci+1}I_{i+1} {}^{i+1}\dot{\omega}_{i+1} + \omega_{i+1} \times {}^{Ci+1}I_{i+1} \omega_{i+1}
\end{aligned} \right\} \quad (4.1)$$

Inward iterations: $i: n \Rightarrow 1$

$$\left. \begin{aligned}
{}^{i+1}f_i &= {}^{i+1}R^i f_{i+1} + {}^iF_i \\
{}^i n_i &= {}^i N_i + {}^{i+1}R^i n_{i+1} + {}^iP_{C_i} \times {}^iF_i + {}^iP_{i+1} \times {}^{i+1}R^i f_{i+1} \\
\tau_i &= {}^i n_i^T \hat{Z}_i
\end{aligned} \right\} \quad (4.2)$$

The gravity effect loading on the links can be incorporated by setting ${}^0\dot{v}_0 = G$, where G has the magnitude of the gravity vector pointing in the opposite direction. The dynamic equations for the two-link manipulator shown in the Figure 15 are computed. For ease of computation, we assume that the mass distribution is simple, and all masses are point masses at the distal end of each link. Since we assume point masses, the inertia tensor at the center of mass of each link will be zero and therefore no forces will be acting on the end-effector.

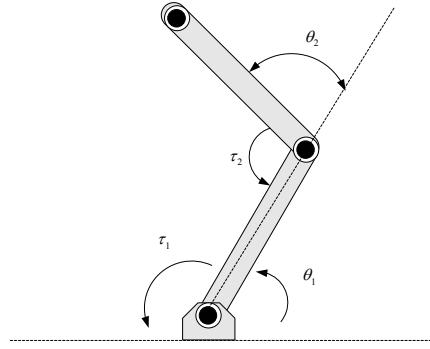


Figure 15: Two-link manipulator with point masses at distal ends of links

Applying equations 4.1 through 4.2, the outward iterations for link1 are as follows:

$${}^1F_1 = \begin{bmatrix} -m_1 l_1 \dot{\theta}_1^2 + m_1 g s_1 \\ m_1 l_1 \ddot{\theta}_1 + m_1 g c_1 \\ 0 \end{bmatrix}$$

$${}^1N_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The outward iterations for link2 are as follows:

$${}^2F_2 = \begin{bmatrix} m_2 l_1 \ddot{\theta}_1 s_2 - m_2 l_1 \dot{\theta}_1^2 c_2 + m_2 g s_{12} - m_2 l_2 (\dot{\theta}_1 + \dot{\theta}_2)^2 \\ m_2 l_1 \ddot{\theta}_1 c_2 - m_2 l_1 \dot{\theta}_1^2 s_2 + m_2 g c_{12} - m_2 l_2 (\ddot{\theta}_1 + \ddot{\theta}_2) \\ 0 \end{bmatrix}$$

$${}^2N_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The inward iterations for link 1 are as follows:

$${}^1f_1 = \begin{bmatrix} c_2 & -s_2 & 0 \\ s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_2 l_1 \ddot{\theta}_1 s_2 - m_2 l_1 \dot{\theta}_1^2 c_2 + m_2 g s_{12} - m_2 l_2 (\dot{\theta}_1 + \dot{\theta}_2)^2 \\ m_2 l_1 \ddot{\theta}_1 c_2 + m_2 l_1 \dot{\theta}_1^2 s_2 + m_2 g c_{12} - m_2 l_2 (\ddot{\theta}_1 + \ddot{\theta}_2)^2 \\ 0 \end{bmatrix} + \begin{bmatrix} -m_1 l_1 \dot{\theta}_1^2 + m_1 g s_1 \\ m_1 l_1 \ddot{\theta}_1 + m_1 g c_1 \\ 0 \end{bmatrix}$$

$${}^1n_1 = \begin{bmatrix} 0 \\ 0 \\ m_2 l_1 l_2 \ddot{\theta}_1 c_2 + m_2 l_1 l_2 \dot{\theta}_1^2 s_2 + m_2 l_2 g c_{12} + m_2 l_2^2 (\ddot{\theta}_1 + \ddot{\theta}_2) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ m_1 l_1^2 \ddot{\theta}_1 + m_1 l_2 g c_1 \end{bmatrix} +$$

$$\begin{bmatrix} 0 \\ 0 \\ m_2 l_1^2 \ddot{\theta}_1 - m_2 l_1 l_2 s_2 (\dot{\theta}_1 + \dot{\theta}_2)^2 + m_2 l_1 g s_2 s_{12} + m_2 l_1 l_2 c_2 (\ddot{\theta}_1 + \ddot{\theta}_2) + m_2 l_1 g c_2 c_{12} \end{bmatrix}$$

The inward iterations for link 2 are as follows:

$${}^2f_2 = {}^2F_2$$

$${}^2n_2 = \begin{bmatrix} 0 \\ 0 \\ m_2 l_1 l_2 \ddot{\theta}_1 c_2 + m_2 l_1 l_2 \dot{\theta}_1^2 s_2 + m_2 l_2 g c_{12} + m_2 l_2^2 (\ddot{\theta}_1 + \ddot{\theta}_2) \end{bmatrix}$$

Extracting the \hat{Z} components of ${}^i n_i$, we find the joint torques:

$$\left. \begin{aligned} \tau_1 &= m_2 l_2^2 (\ddot{\theta}_1 + \ddot{\theta}_2) + m_2 l_1 l_2 c_2 (2\ddot{\theta}_1 + \ddot{\theta}_2) + (m_1 + m_2) l_1^2 \ddot{\theta}_1 - m_2 l_1 l_2 s_2 \dot{\theta}_2^2 \\ &\quad - 2m_2 l_1 l_2 s_2 \dot{\theta}_1 \dot{\theta}_2 + m_2 l_2 g c_{12} + (m_1 + m_2) l_1 g c_1 \\ \tau_2 &= m_2 l_1 l_2 c_2 \ddot{\theta}_1 + m_2 l_1 l_2 s_2 \dot{\theta}_1^2 + m_2 l_2 g c_{12} + m_2 l_2^2 (\ddot{\theta}_1 + \ddot{\theta}_2) \end{aligned} \right\} \quad (4.3)$$

$$\begin{aligned} c_1 &= \cos(\theta_1) & c_2 &= \cos(\theta_2) & s_1 &= \sin(\theta_1) & s_2 &= \sin(\theta_2) \\ c_{12} &= c_1 c_2 - s_1 s_2 & s_{12} &= c_1 s_2 + s_1 c_2 \end{aligned}$$

Equations 4.3 are the torque expressions for the joints which are function of joint position, velocity and acceleration. We can express the dynamic equations in a single

equation that hides some of the details, but shows the structure of the equations. It can be written in the form

$$\tau = M(\Theta)\ddot{\Theta} + V(\Theta, \dot{\Theta}) + G(\Theta) \quad (4.4)$$

Where $M(\Theta)$ is the $n \times n$ mass matrix of the manipulator, $V(\Theta, \dot{\Theta})$ is an $n \times 1$ vector of centrifugal and Coriolis terms, and $G(\Theta)$ is an $n \times 1$ vector of gravity terms [1].

4.5 Manipulator energy-torque relation

A generalized variational problem can be formulated as follows:

$$E\{\theta(t)\} = \min \sum_{t_0}^{t_f} \int_{t_0}^{t_f} F(\theta, \dot{\theta}, \ddot{\theta}, t) dt$$

where the integrand function $F(\theta, \dot{\theta}, \ddot{\theta}, t)$ is the object of optimization, and it may represent any function of time or robot coordinate parameters.

θ is an m-DOF position vector in joint space and $\dot{\theta}$ and $\ddot{\theta}$ are its derivatives. The boundary conditions of the integral are defined according to the movement characteristics at the initial and final state of the robot manipulator.

$$\dot{\theta}(t_0) = \dot{\theta}(t_f) = \ddot{\theta}(t_0) = \ddot{\theta}(t_f) = 0$$

The initial and final values of the velocity and accelerations are zero, so that the manipulator is standing still at the starting and final points.

The integrand $F(\theta, \dot{\theta}, \ddot{\theta}, t)$ represents the total electrical energy of the robot manipulator. If DC motors are used as the actuators for the joints, the total electrical energy of the robot manipulator is calculated as follows [21]:

$$E_t = \sum_{j=1}^m \int_{t_0}^{t_f} u_j i_{aj} dt \quad (4.5)$$

where u_j and i_{aj} are the terminal voltage and current of the motor of the j^{th} axis, and m is the total number of axes. The electrical equations of the DC motor drive at the j^{th} robot axis are written as follows

$$\left. \begin{aligned} u_j &= R_{aj} i_{aj} + L_{aj} \frac{d}{dt} i_{aj} + E_{aj} \\ \tau_j &= K_{tj} i_{aj} \\ E_{aj} &= K_{ej} \frac{d}{dt} \theta_j \end{aligned} \right\} \quad (4.6)$$

The general dynamic equation of the robot manipulator from equation 4.4 is

$$\tau = M(\Theta)\ddot{\Theta} + V(\Theta, \dot{\Theta}) + G(\Theta) \quad (4.7)$$

Substituting 4.6 through 4.7 in 4.5 and grouping the terms yields the following equation for the total electrical energy.

$$E_t = \int_{t_0}^{t_f} [\ddot{\theta}^T A(\theta)\ddot{\theta} + \dot{\theta}^T B(\theta)\dot{\theta} + \dot{\theta}^T C(\theta)\dot{\theta} + D(\theta)\ddot{\theta} + E(\theta)\dot{\theta} + F(\theta)] dt \quad (4.8)$$

where

$$\begin{aligned}
A(\theta) &= R_a K J(\theta)^T J(\theta) \\
B(\theta) &= R_a K H(\theta) J(\theta) + Ke Kt^{-1} J(\theta) \\
C(\theta) &= (R_a K H(\theta)^T + Ke Kt^{-1}) H(\theta) \\
D(\theta) &= R_a K J(\theta) G(\theta) \\
E(\theta) &= (R_a K H(\theta) + Ke Kt^{-1}) G(\theta) \\
F(\theta) &= G(\theta)^T R_a K G(\theta) \\
K &= (Kt Kt^T)^{-1} \\
Ra &= \text{diag}(Ra_1, Ra_2, \dots, Ra_m) \\
Kt &= \text{diag}(Kt_1, Kt_2 \dots Kt_m) \\
Ke &= \text{diag}(Ke_1, Ke_2, \dots, Ke_m)
\end{aligned}$$

4.5.1 Gravity contribution in the energy torque equation

In the energy-torque equation the gravity effect is represented by the function $G(\Theta)$. Based on many simulation studies using different trajectories, it was found that the effect of the gravity term was negligible, and the energy loss with the gravity term was poor for all the cases. In the case of the articulated two-link manipulator mechanical characteristics, the friction parameters have a greater effect than the weight parameter of each arm of the robot. Another reason is that if the gravity term is included, the minimization process also tends to minimize the potential energy of the manipulator.

As a result, the kinetic energy in the minimization equation will be decreased.

Neglecting $G(\Theta)$ yields

$$\ddot{\theta}^T A(\theta) \ddot{\theta} + \dot{\theta}^T B(\theta) \dot{\theta} + \dot{\theta}^T C(\theta) \dot{\theta} \quad (4.9)$$

The matrices $A(\theta)$, $B(\theta)$, $C(\theta)$ are not diagonal or constant. From the perspective of the stability and simplicity of the iterative calculation, diagonal and time-invariant A ,

B , and C matrices would be advantageous. It is found that the simplification of Equation 4.9 as shown in Equation 4.10 results in almost the same minimization of the consumed electrical energy with much less computation time. Thus the equation below is proposed for the minimal energy relation.

$$\sum_{i=1}^m A_i \ddot{\theta}_i^2 + B_i \dot{\theta}_i \ddot{\theta}_i + C_i \dot{\theta}_i^2 \quad (4.10)$$

The coefficients A_i , B_i , and C_i are constant. Equation 4.10 will be used as the cost function for the optimization process for trajectory planning of the manipulator.

CHAPTER V

EVOLUTIONARY ALGORITHMS

Evolutionary algorithms are considered as a broad class of stochastic optimization techniques motivated by the process of natural evolution found in biological organisms. The term *evolutionary algorithm* has been introduced in newer literature as a generic term to denote a family of popular algorithms, which have been developed independently since the beginning of the 1960s. Essentially these are: Genetic Algorithms, Evolutionary Programming and Evolutionary Strategies. All of them share the same basic structure; however they differ in details and especially in their history of development. The basic approach for optimization is to devise a single standard of measurement-a cost function that summarizes the performance or value of a decision and iteratively improve this performance by selecting from among the available alternatives [22].

We address Genetic Algorithms as an optimization tool for minimization of electrical energy consumed in manipulators. In section 5.1 we present the introduction to genetic algorithms and also the common GA terms. Then we discuss the basic components of GA in section 5.2. In section 5.3 we explain the GA approach to manipulator trajectory path and the problem formulation for this work.

5.1 Introduction to genetic algorithms

Genetic algorithms (GAs) are well-known evolutionary algorithms (EAs), receiving a great attention all over the world. They were developed by Holland [23] and are based on the genetic processes of biological organisms. GAs use a direct analogy of natural behavior. GAs operate on a population of points, assigned as individuals. Each individual of the population can be a possible solution of the optimization problem. Individuals are evaluated depending upon their fitness.

The GA's initial populations of strings are generated at random. By the application of genetic operators, selection, crossover, and mutation, the transition of one population to the next population takes place.

In the selection process, the fittest individuals will be preferred to go to the next generation. The crossover operator exchanges the genetic material of two individuals, creating two new individuals. Mutation changes the genetic material of an individual. The least fit members of the population are less likely to get selected for reproduction, and so die out. The application of the genetic operators upon the individuals of the population continues until a sufficiently good solution of the optimization problem is found. The process of genetic computation continues until it reaches a predefined stop condition, i.e., until a certain number of generations is reached.

GAs operate with a population of strings instead of a single individual. Thus, the search is carried out in a parallel form. They inspect many possible solutions at the same time. So there is a higher probability that the search converges to an optimal solution. GAs are able to find optimal solutions in intricate and large search spaces. In addition,

GAs are appropriate for nonlinear optimization problems that can be defined in discrete or continuous search spaces.

In the traditional algorithms, GA strings are represented by binary numbers. Now with the developments in the GA field, new representations for the individuals have been developed. The real representation has shown to be more appropriate for optimization problems with the variables within a continuous domain. In the present work, we employ the real representation for the individual. The following properties make genetic algorithms attractive: they are simple, robust, effective, and less tendency to converge to local optima, as compared to gradient-based search methods [7].

5.1.1 Common GA terms

We present the common GA terms in this section. Since a GA tries to mimic the behavior of a process of natural evolution, the terminology would be similar but not identical to the terms in natural genetics. Population is the significant term for a genetic algorithm. A population P consists of individuals C_i with $i=1, \dots, \lambda$:

$$P = \{C_1, \dots, C_i, \dots, C_\lambda\}.$$

The cost function $g(x)$ of the optimization problem is a scalar-valued function of an n -dimensional vector x . The vector x consists of n variables x_j with $j = 1 \dots n$, which represents a point in real space \mathfrak{R}^n . The variables x_j are called genes. Thus, an individual C_i consists of n genes:

$$C_i = [C_{i1}, \dots, C_{ij}, \dots, C_{in}]$$

In the classical GA, the individuals were represented as binary numbers. In this case, binary coding and gray coding can be used.

Fitness of an individual is another principal term in genetic algorithms. In a GA, each individual represents one point in the search space. The individuals undergo a simulated evolutionary process. In each generation, relatively good-fit individuals reproduce, while less-fit individuals do not survive. The fitness of an individual serves to differentiate between the individuals. The fitness of an individual is calculated by the fitness function $F(x)$. For optimization problems without constraints, the fitness function depends on the objective function of the optimization problem on hand.

For maximization problems $F(x) = g(x)$, and for minimization problems $F(x) = -g(x)$, where $g(x)$ is the objective function. The electrical energy consumed by the manipulator is the objective function in this work. Since we are interested in the minimal energy path, the objective function will be negative.

In general, genetic algorithms start with a random initialization of individuals, and the fitness for each individual is evaluated. Then the genetic operators *selection*, *crossover* and *mutation* are applied. Thus new individuals are produced from this optimization process, which then results in the next population.

Another common term is generation, which is the transition of an old population P_g to a new population P_{g+1} , where g designates the generation number. In Figure 16, the process of optimization executed during a generation is depicted. This continues for quite a few generations until the problem is solved, which in most cases ends in a maximum number of generations g_{max} [24].

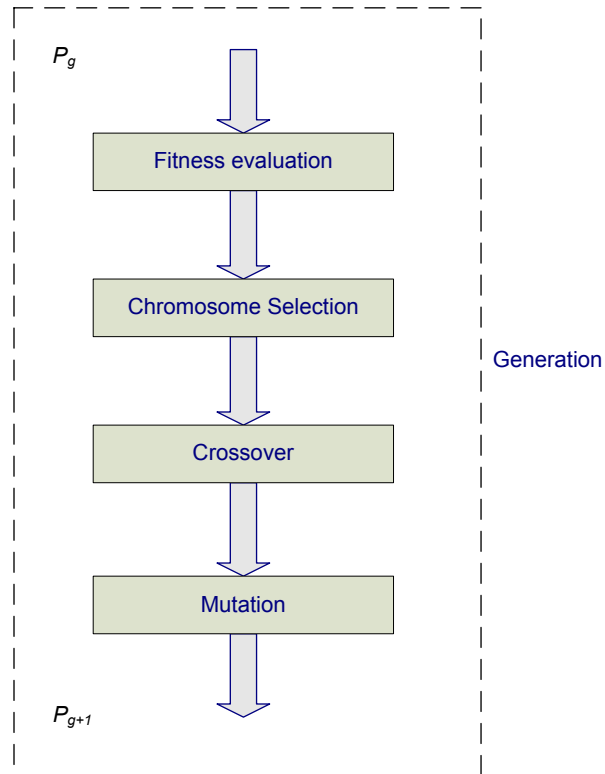


Figure 16: Executed operations during a generation

The chromosome of the population looks like the one in Figure 17.

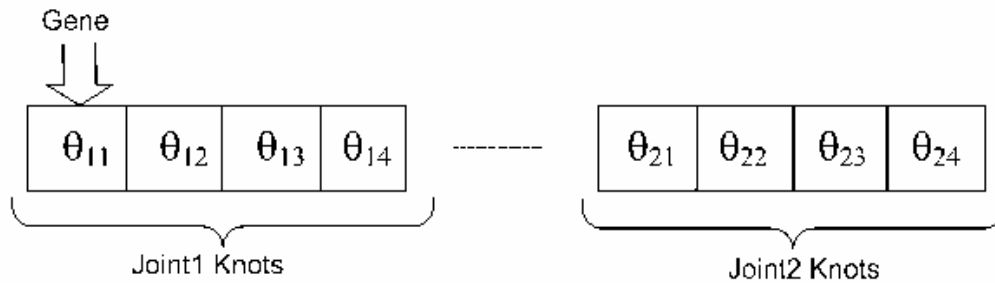


Figure 17: Representation of an individual

In our case of a two-link manipulator, the intermediate knots for the two joints form a chromosome.

5.2 Basic components of a genetic algorithm

By the application of a genetic operator's *selection*, *crossover*, and *mutation* processes, a GA transition from one generation to another generation takes place. The following sections describe the three components of a genetic algorithm.

5.2.1 Selection

The selection process chooses the fittest individuals from a population to continue into the next generation. Based on Darwin's principle of natural selection, "survival of the fittest," the selection component chooses the fittest individuals. It can be a deterministic operation, but in most implementations it has random components.

The selection component judges against the fitness of one individual in relation to other individuals and chooses which individual goes on to next generation. Through selection, "good individuals" are favored to advance with a high probability, while "bad individuals" advance with low probability to the next generation. The following sections describe the different selection methods.

Proportionate selection method

The thought behind the proportionate selection method is that each individual is given a chance to become a parent in proportion to its fitness. It is also called roulette-wheel selection, as the chances of selecting a parent can be seen as spinning a roulette wheel, with the size of the slot for each parent being proportional to its fitness. Apparently, those with the larger slot sizes have a greater chance of being selected.

Therefore, it is possible for one member to govern all the others and get selected a high proportion of the time.

Tournament selection

In this choice of selection, a predefined number of individuals are chosen randomly from the population, and the best individual from this group is selected as a parent which advances to the next population. This process is repeated as often as individuals must be chosen. These selected parents produce uniform random offspring. The parameter for tournament selection is the tournament size z . The number z ranges from two to the number of individuals in the population. In many applications of this method, the tournament is carried out only between two individuals and is called a binary tournament selection.

There are two different types of tournament selection:

- **Strict tournament**

Here the fittest individual from a group of z individuals (Tournament size) will be selected for crossover. If the fit individual is one of those z individuals, the probability of that individual being selected for the crossover will be one. If a median fit is one of those z individuals, then the probability of it getting selected can be given as $(\frac{1}{2})^{z-1}$. Thus, the selection pressure for a tournament selection, which is the probability that the fittest individual is selected as a parent relative to an individual with average fitness, is always greater than two. The selection pressure can be easily increased by increasing the tournament size.

- **Soft Tournament**

In this method z individuals are picked randomly, and the most fit among them is selected for crossover, similar to the strict tournament. But in this case, the rest of the population will also be picked with the remaining probability. The optimizing performance is improved by this method as it decreases the selection pressure.

5.2.2 Crossover

In the selection process, only copies of individuals are inserted into the new population. A crossover is a genetic operator that mates two chromosomes to produce a new chromosome. The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability. Crossover in natural genetics is described as the process of producing new genetic material by exchanging the genetic material between the individuals of the population. As a result, new individuals are produced. Two individuals are chosen and crossed. The resulting offspring replace the parents in the new population. Crossover can lead to the loss of “good” genetic material. Therefore, the crossover of two individuals is carried out with the probability P_c , the crossover probability, which is fixed before the optimization process.

A random number uniformly distributed between zero and one is generated. If this number is smaller than P_c , then two individuals of the population randomly chosen are split at a crossover point. The crossover point determines how the genetic material of the

new individuals will be composed. For each pair of individuals, the crossover point is randomly determined. The representation determines how the crossover operator is applied to the individuals. The crossover for binary and real representation is described below. .

Crossover for binary representation

In this representation, the individuals of the population are binary numbers, and the crossover is performed as follows.

Consider the following two parents which have been selected for crossover. Next, they are divided at the chosen crossover point j into two parts and are newly composed.

Old parents:

$$C_1 = [C_{1,1}, C_{1,2}, \dots, C_{1,j+1}, \dots, C_{1,n}]$$

$$C_2 = [C_{2,1}, C_{2,2}, \dots, C_{2,j+1}, \dots, C_{2,n}]$$

After the crossover operation is done at the crossover point j , two new offspring's would be:

$$C_1^{new} = [C_{1,1}, C_{1,2}, \dots, C_{1,i}, C_{2,j+1}, \dots, C_{2,n}]$$

$$C_2^{new} = [C_{2,1}, C_{2,2}, \dots, C_{2,i}, C_{1,j+1}, \dots, C_{1,n}]$$

The new offspring consists of the first part of the old individual C_1 and the second part of the old individual C_2 . Similarly, the second new offspring consists of the first part of the old individual C_2 and the second part of the old individual C_1 . Figure 18 shows an example.

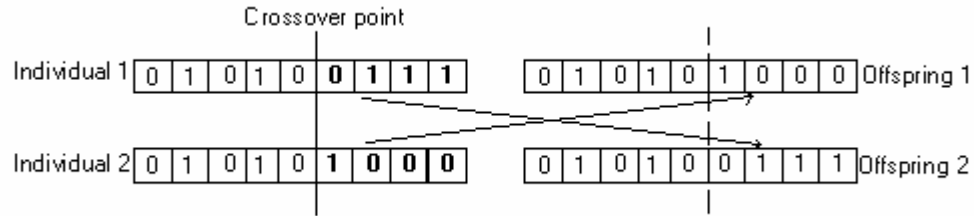


Figure 18: Crossover for binary representation

The description above shows a one-point crossover. For a two-point crossover the operator randomly selects two crossover points within a chromosome, then interchanges the two parent chromosomes between these points to produce two new offspring.

Consider the following two parents which have been selected for crossover. The “|” symbols indicate the randomly chosen crossover points.

Parent 1: 110|010|10

Parent 2: 001|001|11

After interchanging the parent chromosomes between the crossover points, the following offspring are produced:

Offspring1: 110|001|10

Offspring2: 001|010|11

Another kind of binary crossover operator is the uniform crossover. With some probability, usually known as mixing ratio, a uniform crossover operator decides which parent will contribute each of the gene values in the offspring chromosomes. This allows the parent chromosomes to be mixed at the gene level rather than the segment level (as with one- and two-point crossovers). For some problems, this additional flexibility outweighs the disadvantage of destroying building blocks.

Consider the following two parents which have been selected for crossover:

Parent1: 01000011

Parent2: 00100111

If the mixing ratio is 0.5, approximately half of the genes in the offspring will come from parent 1 and the other half will come from parent 2. Below is a possible set of offspring after a uniform crossover. The subscripts indicate which parent the gene came from.

Offspring 1: $0_1 0_2 1_2 1_1 0_2 0_1 1_1 1_2$

Offspring 2: $0_1 0_2 0_2 1_1 0_1 0_1 1_2 1_2$

Crossover for real representation

In a real representation the individual is a real number. The crossover operator linearly combines two parent chromosome vectors to produce two new offspring according to the following equations.

$$\text{Offspring1} = a * \text{Parent1} + (1 - a) * \text{Parent2}$$

$$\text{Offspring2} = (1 - a) * \text{Parent1} + a * \text{Parent2}$$

where a is a random weighting factor (chosen before each crossover operation).

In the case of trajectory planning for a manipulator, if we consider that we have four knots for each joint, then the chromosome would have 8 genes in the individual.

The following two parents have been selected for crossover:

Parent1: {4.6782, 2.7641, 4.2935, 5.2731, 1.6836, 5.8646, 1.3356, 3.9508}

Parent2: {3.3904, 4.3096, 5.5093, 1.950, 3.9169, 4.2559, 0.0810, 4.8951}

If $a = 0.7$, the following two offspring would be produced:

Offspring1: {4.2919, 3.2278, 4.6582, 4.2763, 2.3536, 5.3820, 0.9592, 4.2341}

Offspring2: {3.7768, 3.8459, 5.1446, 2.9471, 3.2469, 4.7385, 0.4574, 4.6118}

5.2.3 Mutation

A mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at a better solution than was previously possible. A mutation is an important part of the genetic search as it helps to prevent the population from stagnating at any local optima. A mutation occurs during evolution according to a user-definable mutation probability.

A mutation is executed with the probability P_m , *the mutation probability*, which is fixed before the optimization. For each individual, a random number between 0 and 1 is calculated, which is compared with the mutation probability. If the random number is smaller than the probability of mutation, a gene is mutated.

In binary representation, one bit of a gene is selected randomly at a bit position and inverted; i.e., a bit with the value 0 is replaced with a bit with the value 1, and vice versa. See Figure 19.

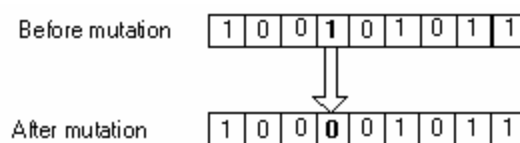


Figure 19: Binary mutation

The mutation operator was originally developed for binary representation. Since then other methods have been developed that allow for gene modification in real representation.

5.3 Genetic algorithms approach to the manipulator trajectory problem

In a genetic algorithm, a population of strings is processed many times. Each member string in the population represents a possible solution. Some strings represent infeasible solutions, while some represent a good solution. Eventually after much processing, the population converges to the best possible solution, i.e., only the copies of good solutions are left and the bad ones are eliminated.

In our case of a two degree-of-freedom manipulator, a string would represent the knots which are the intermediate points in the trajectory path for each joint. The best string or chromosome is the one which would optimize the total electrical energy consumed in the manipulator.

The chromosome of the population looks like the one in Figure 17. Since here we are directly coding the string of real numbers, the process would be called a "*Real-coded Genetic Algorithm*".

Genetic algorithms start with an initial population of individuals. The population is arbitrarily initialized within the joint bounds. The processes of selection, crossover and mutation help the population evolve towards better and better regions in the search space.

Problem representation

Let it be assumed that a two-link planar robotic manipulator has to move from an initial position to a fixed final position. The objective is to find an optimal path that takes minimum electrical energy.

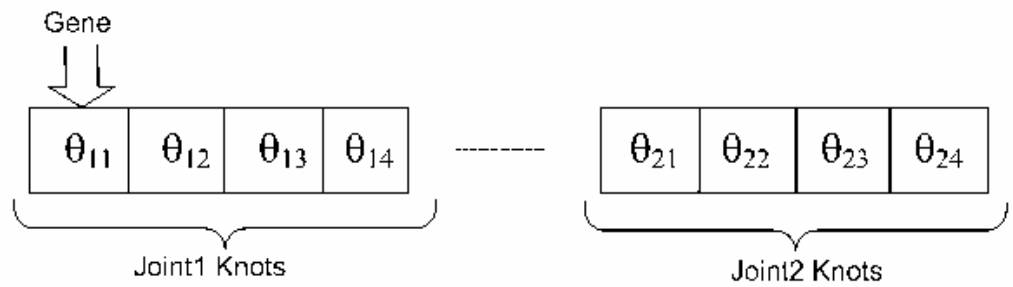
The following assumptions are made to simplify the problem:

1. The robot is considered to be a two-link planar manipulator.
2. Kinematic constraints that limit the robotic manipulator are as follows: The trajectory points should be defined in its dexterous workspace, and the method of solution, whether a left-or right-arm system should be defined. A Boolean flag is used in the design code to differentiate the left arm and right arm system.
3. The trajectory time is defined along with the initial and final Cartesian points.
4. The manipulator total path has intermediate points called knots which are obtained from the genetic algorithm and are considered at regular intervals of time.
5. The cubic spline approach is employed for the individual paths between the knots.
6. It is assumed that the end-effector of the manipulator starts from zero velocity and ends at zero velocity and doesn't stop at the intermediate knots.

With the above assumptions, the whole design process for optimal trajectory planning is summarized as follows:

1. In the first step, the link lengths, trajectory time, number of knots, Cartesian positions (initial and final), population size, maximum number of generations, and the crossover parameter are defined.

2. The Cartesian positions are transformed to joint positions using inverse kinematics. Inverse kinematics is applied only for the initial and final positions and not for the intermediate points. Right arm or left arm solutions should be marked out at this stage. Forward kinematics validates the transformations and is done using the SimMechanics model.
3. The chromosome for the random population looks like the one below.



4. Each joint trajectory is spline-interpolated using the *cubic spline approach*, and the path generator computes the position, velocity and acceleration as t is advanced. When the end of one segment is reached, a new set of cubic coefficients is recalled, t is set back to zero, and the generation continues.
5. Fitness is evaluated for each chromosome in terms of the consumed electrical energy. The energy relation for the manipulator is based on its link lengths, inertia tensor, friction constant, armature resistance, instantaneous velocity and acceleration. The equation below is proposed for the energy relation:

$$\sum_{i=1}^m A_i \ddot{\theta}_i^2 + B_i \dot{\theta}_i \ddot{\theta}_i + C_i \dot{\theta}_i^2$$

The derivation for this relation is discussed in the previous chapter.

6. The genetic operators are applied upon the individuals of the population until a sufficiently optimum fitness value is found. The solution is achieved at a predefined stop condition, i.e., after a maximum number of generations is reached.

The next chapter presents the simulation results for different GA runs for spline interpolation trajectories, linear interpolated trajectories and also on SimMechanics models.

CHAPTER VI

RESULTS AND CONCLUSIONS

In this chapter, we present the simulation work for optimal trajectory planning for a two link industrial manipulator system. The minimum consumed energy is used as a criterion for trajectory generation, by using genetic algorithm as an optimization tool. We propose the use of cubic spline curve to generate the trajectory between the initial and final positions of the end effector. All the simulation work is done in MATLAB.

Section 6.1 presents the results for different trajectory schemes in which we discuss the joint and Cartesian space schemes. Then we present the GA results for the optimal energy in section 6.2 for a specific geometrical problem. Section 6.3 demonstrates the SimMechanics model and explains how it is used for trajectory problems. The final section ends with some concluding remarks and discusses the possibilities for further development and improvement of the proposed approach.

6.1 Trajectory analysis

Initially in this chapter we present the simulation results for three different methods of trajectory generation between initial and final position of the trajectory path.

In Section 6.1.1 we present the cubic spline approach results for path planning for manipulators which is the proposed method in this work. As we discussed before, the cubic spline polynomials are smooth and continuous across the interval and would be ideal to use for optimal path generation. We give the user the capability of specifying the number of knots between the initial and goal Cartesian coordinates. In this work, we consider that the knots are at regular intervals of time, generated randomly.

In section 6.1.2 we present another joint scheme of path generation where the linear move is between the initial and final point in the joint scheme. Since the joints have to be coordinated, which means they start at the same time and end at the same time, the individual joint paths will not be linear because the linear distance will not be the same for all joints. This scheme is usually easy to compute and there is essentially no problem with singularities.

In section 6.1.3 we discuss the Cartesian straight line path generation results. In this trajectory scheme the linear path is considered in the Cartesian space. The linear interpolation algorithm is almost the same for the linear joint space scheme and the Cartesian space. The end-effector in this case follows a straight line. In a system where Cartesian paths are allowed, path shapes such as circles, ellipses, and sinusoids can also be executed by closely defining the Cartesian points. These paths have the disadvantage of singularities in the workspace.

The method of solution considered here is a left arm solution. Some of the simulation parameters which are consistent for all the three trajectory paths are shown below.

Initial Cartesian point = [50 30]

Final Cartesian point = [-50 30]

Trajectory time = 6 seconds

Link1 length = 50 centimeters

Link2 length = 50 centimeters

6.1.1 Cubic spline polynomials with via points

In this method of trajectory generation we consider intermediate knots between the initial and final positions of the path. The joint positions for the corresponding Cartesian points are obtained from the inverse kinematics.

Left arm solution:

Initial joint point = $[J1_{\text{initial}} \ J2_{\text{initial}}] = [1.4887 \ -1.8965]$ radians

Final joint point = $[J1_{\text{final}} \ J2_{\text{final}}] = [3.5494 \ -1.8965]$ radians

Number of knots = 5 between initial and final knot positions

Joint1 knots = [1.4887 4.5686 5.2684 2.3274 3.4342 4.3641 3.5494]

Joint2 knots = [-1.8965 1.9433 3.5693 4.4154 2.7953 3.9038 -1.8965]

The knot values are randomly chosen values. The constraints for this trajectory path are that the initial and final velocities are zero, and the intermediate knots are at regular intervals of time. The simulation results for the joint positions and velocities are shown in Figure 20 and Figure 21.

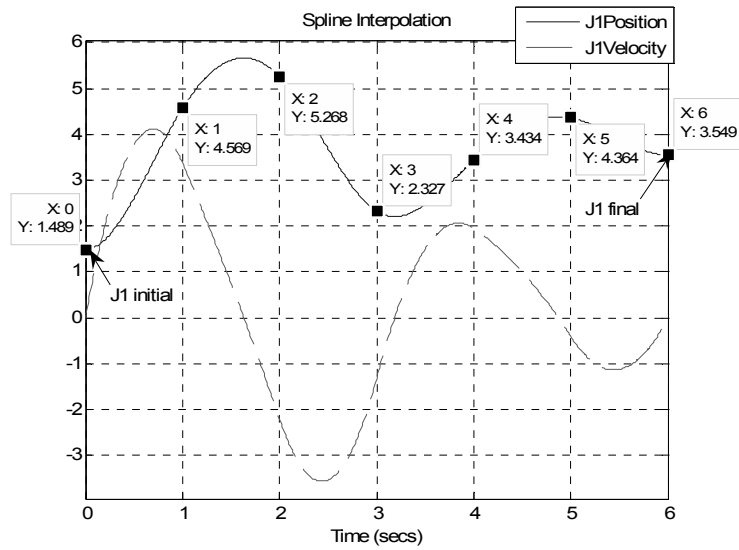


Figure 20: Joint1 position (radians) and velocity(radians/sec)

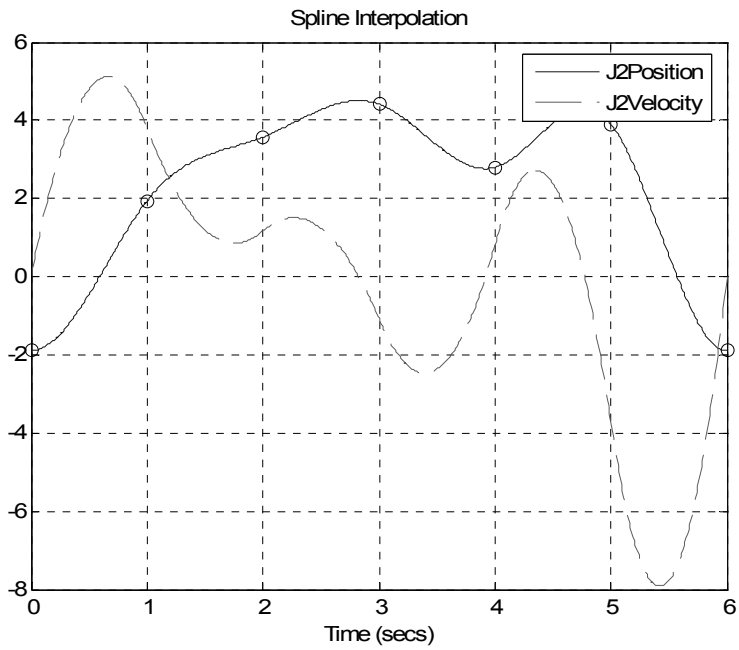


Figure 21: Joint 2 position(radians) and velocity(radians/sec)

As we described before, there can be multiple paths between initial and final positions. Figure 22 shows the different paths generated between initial and final joint

positions. There are five intermediate knots between the initial and final knot positions.

The intermediate knot positions are generated randomly and connected by splines.

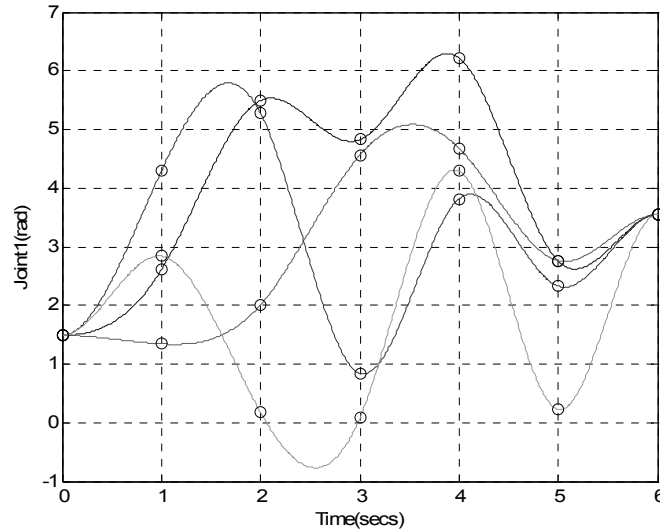


Figure 22: Joint1 knots at different positions

6.1.2 Linear path in joint space

Another choice of trajectory generation is the linear path in joint space. The additional simulation parameters for this trajectory path are the acceleration time and deceleration time for the move. V_{\max} is calculated for different acceleration and deceleration times.

Left arm solution:

$$\text{Initial Joint point} = [J1_{\text{initial}} \ J2_{\text{initial}}] = [1.4887 \ -1.8965] \text{ radians}$$

$$\text{Final Joint point} = [J1_{\text{final}} \ J2_{\text{final}}] = [3.5494 \ -1.8965] \text{ radians}$$

Ta = Acceleration time

Td = Deceleration time

$$Tc = \text{Constant velocity time} = \text{Trajectory time} - Ta - Td$$

V_{\max} = Maximum velocity for the move

TABLE I: VELOCITY COMPUTED FOR DIFFERENT INTERVALS IN JOINT SPACE

Ta (sec)	Td (sec)	Tc (sec)	V_{\max} (rad/sec)
2	1	3	0.4579
3	3	0	0.6869
1	4	1	0.5828

The joint1 position over the time of 6 seconds is shown in Figure 23 for different acceleration and deceleration times. The boxes in the figure represent Ta and Td. As described before, the joint paths are not linear since both of the joints are coordinated and vary based on the acceleration and deceleration times as shown below.

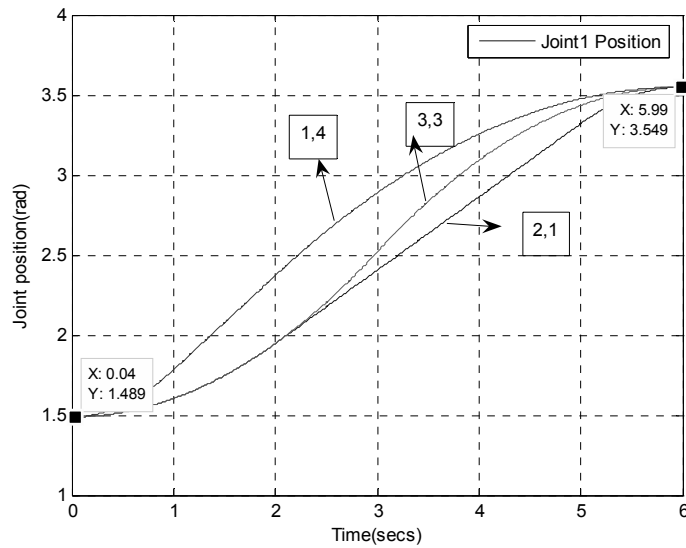


Figure 23: Joint 1 (radians) trajectory

The Joint 2 initial and final positions are the same for the whole trajectory in this problem. So it would be a linear path with respect to time. The joint velocities and the vector velocity for the move are shown in Figure 24.

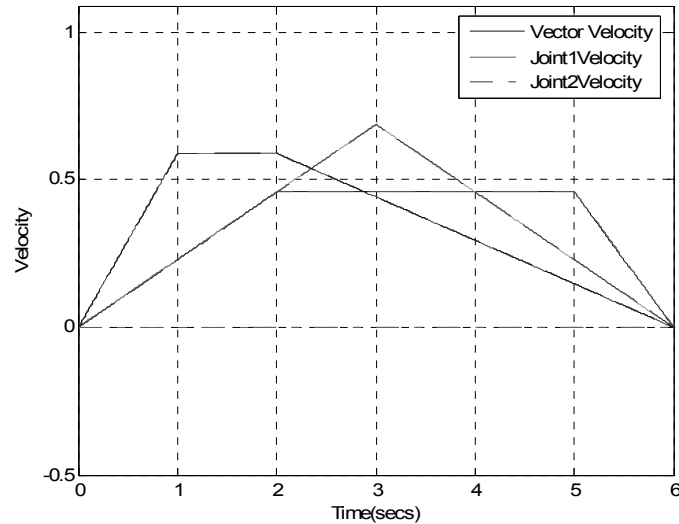


Figure 24: Joint velocities for different acceleration times

Irrespective of the acceleration and deceleration times, the Cartesian path for the whole move would be the same as the dotted line shown in Figure 25.

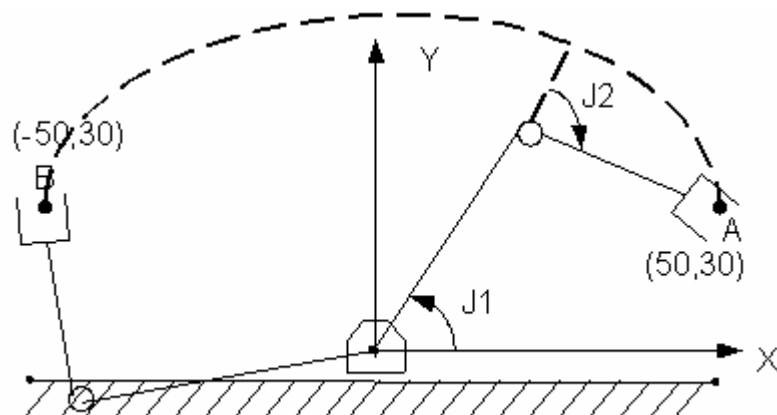


Figure 25: Cartesian trajectory path

6.1.3 Cartesian straight-line motion

Sometimes, we might need to specify a spatial path that causes the end-effector to move through space in a straight line. Obviously, if we specify or generate many closely separated via points lying on a straight line, the tool tip will appear to follow a straight line as shown in Figure 26. The result of trajectory path from any of the above trajectory methods is a set of data for each segment of the trajectory. Here in this case the interpolator generates the data points at the path update rate, and the inverse kinematics generates the corresponding joint positions. The velocity profile and trajectory deviation are dependent on the path update period, and the velocity profile will be smooth if it is very small.

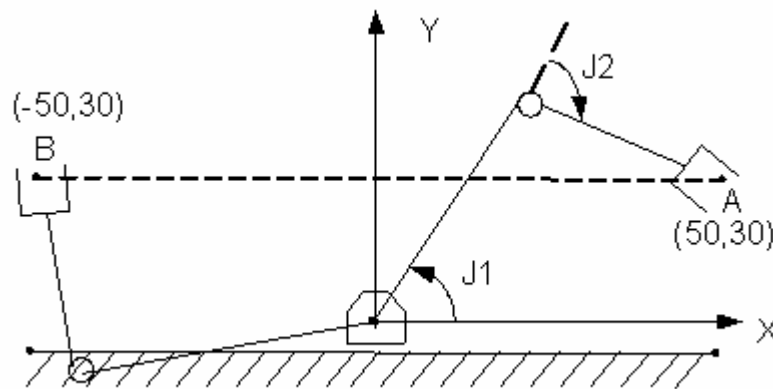


Figure 26: Manipulator Cartesian path

V_{\max} is computed for different acceleration and deceleration times.

TABLE II: VELOCITY COMPUTED FOR DIFFERENT INTERVALS IN CARTESIAN SPACE

T_a (sec)	T_d (sec)	T_c (sec)	V_{max} (cm/sec)
2	1	3	22.22
3	3	0	33.33
1	4	1	28.57

The end-effector has the vector velocity paths for different acceleration and deceleration times as shown in Figure 27.

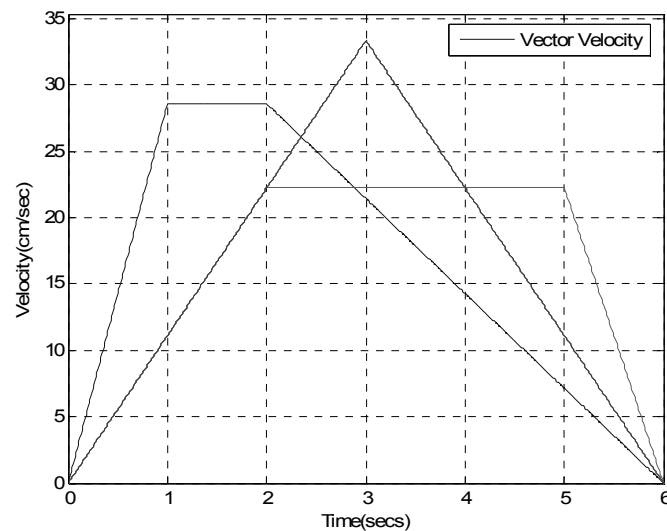


Figure 27: Vector velocity of the Cartesian move

Figure 28 shows the joint trajectories for a Cartesian straight line. We can see that for joint 2, even though the initial and final joint positions are the same, the joint trajectory followed a curved path so that the tip of the tool follows a Cartesian straight line.

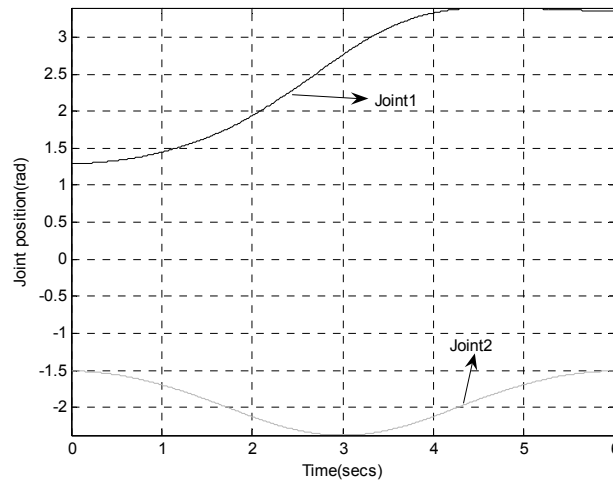


Figure 28: Trajectory of each joint

6.2 Genetic approach to trajectory generation

The GA approach for trajectory generation was discussed in section 5.3. The initial population is generated randomly as per the problem-specific constraints (for example, joint range limits).

We have seen that in cubic spline trajectory generation, the intermediate knots can be anywhere between the initial and final positions. The chromosome of the population consists of the knots for each joint. We evaluate the fitness of each chromosome and apply the genetic operators until a sufficiently optimum fitness value is found. The chromosome corresponding to the minimum consumed energy will have the information for the whole trajectory of the joints.

The same initial and final points as discussed in the previous section are considered for the first simulations.

Spline interpolation results

In the simulation we use the following genetic parameters.

Population size = 50

Chromosome Length = 6

Maximum number of generations = 30

Crossover parameter = 0.7

We use the real representation, so the chromosome length is six, which is two times the number of knots. The above parameters were obtained by performing simulations with different values for the genetic parameters and running a GA. The sensitivity of the genetic algorithm is not much affected by the choice of parameters. As the number of knots increases, the chromosome length increases, which affects the GA computation. The selection scheme employed here is the roulette wheel selection. At the end of 20 generations, the maximum fitness and mean fitness are almost the same. This shows the convergence of the population to the least energy. The best individual trajectories are found to be

Best Joint1 = [1.4887 3.0743 3.5742 4.0723 3.5494]

Best Joint2 = [-1.8965 4.1894 2.7414 2.8486 -1.8965]

Figure 29 shows the trajectories of the best individuals. The total energy consumption or the fitness evaluated for this joint angles set is found to be **530.45 Joules**.

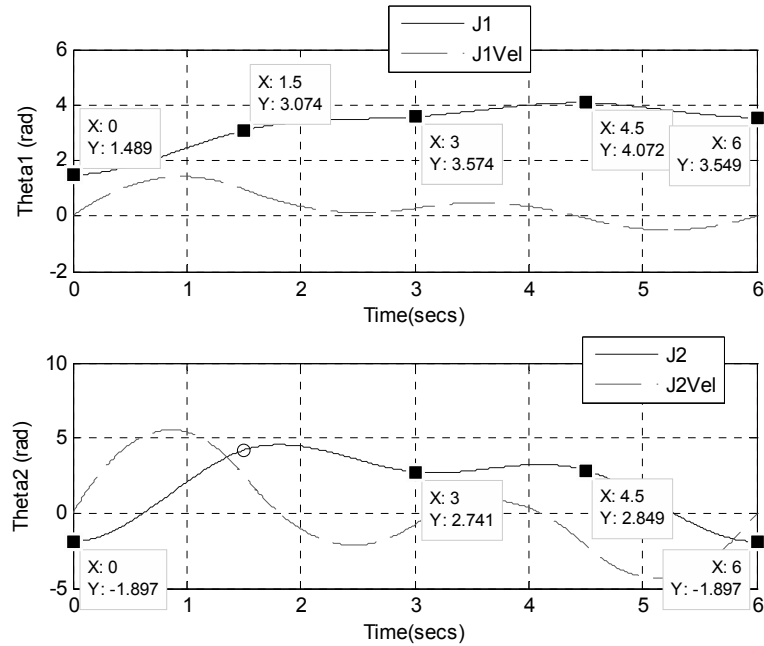


Figure 29: Best individual trajectories

For minimization problems, it is essential to change the objective function, because the GA works according to the principle of maximization of the fitness [24]. By multiplying the objective function with a factor of negative one, the minimization problem is transformed into a maximization problem. Figure 30 shows the mean and minimum energy curves for 30 generations. The minimum energy plot shows the convergence of the GA at generation 10.

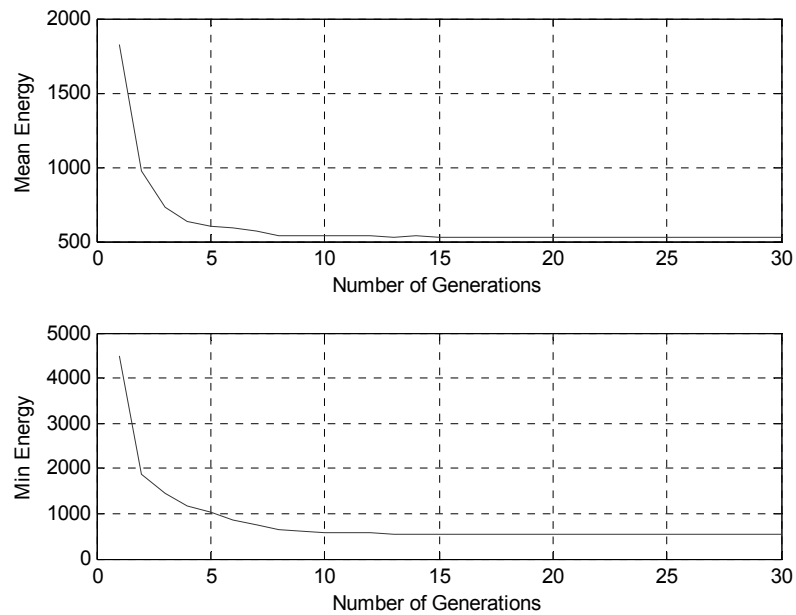


Figure 30: Mean fitness and maximum fitness

Cartesian straight line results

In the Cartesian straight line trajectory scheme, with acceleration time 2 seconds and deceleration time 2 seconds, we generate the path that causes the tip to follow a straight line. The trajectory path for the joints is as shown in Figure 28.

The instantaneous joint positions, joint velocities and accelerations at the path update period are calculated, and the energy consumption is evaluated. At each iteration period, joint position is computed using inverse kinematics, whereas in joint schemes the joint position is calculated only for initial and final points. The total energy consumption for the manipulator for this case was found to be **2895.2 Joules**. Looking at Figure 28 it appears that a good solution would be to leave joint 2 nearly constant.

Figure 31 shows the case of interpolation, where joint 2 nearly remain constant. We choose a trapezoidal velocity profile along the complete path to produce a joint trajectory. The end-effector path will not be a straight line. The energy consumed in this case is **2178.5 Joules**.

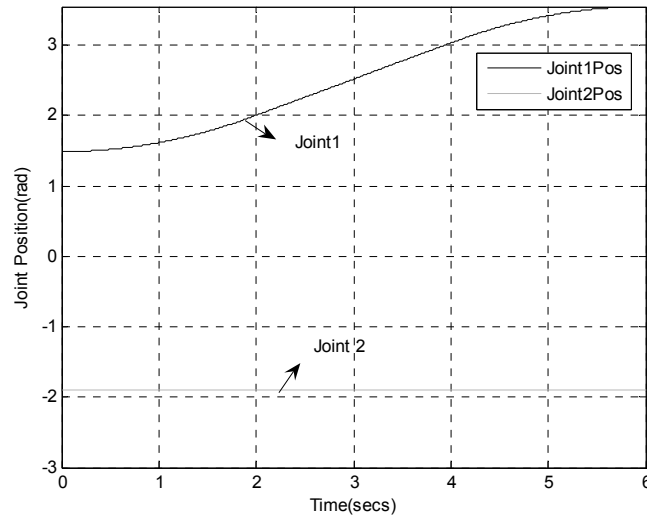


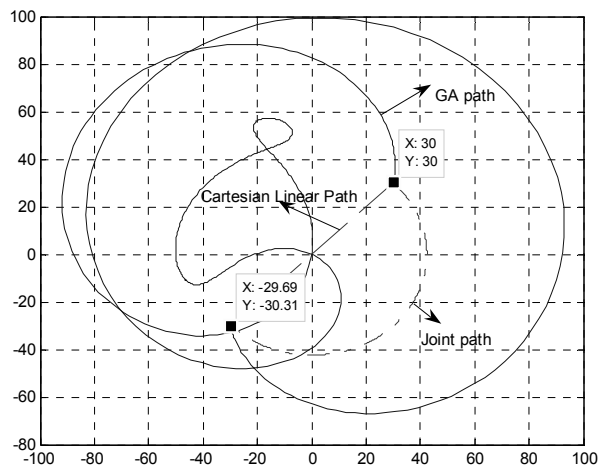
Figure 31: Joint 2 nearly constant

The following are few simulation results with different initial and final Cartesian positions. The first simulation has high energy consumption. This is because the manipulator is passing through the origin. Initially, we define the method of solution to be a left arm solution. There would be a high joint velocity at this time.

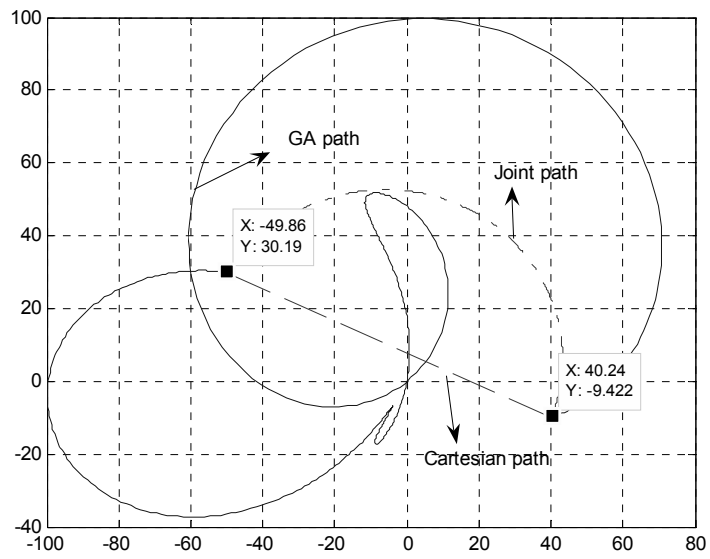
TABLE III: ENERGY COMPUTED FOR DIFFERENT INITIAL AND FINAL POINTS

No	Cartesian Space	Joint Space	GA path Energy (Joules)	Cartesian linear path Energy (Joules)	Joint linear path Energy (Joules)
1	Initial: (30,30) Final: (-30,-30)	Initial: (1.91, -2.26) Final: (-1.22, -2.26)	881.32	7452.6	5063.1
2	Initial: (40, -10) Final: (-50, 30)	Initial: (0.90,-2.29) Final: (3.54, -1.89)	754.29	2760.0	3678.8
3	Initial: (-50, -60) Final: (50, -30)	Initial: (-1.59, -1.34) Final: (0.40, -1.89)	720.35	453.92	2203.5
4	Initial: (40, -5) Final: (-50, -8)	Initial: (-1.03,-2.31) Final: (-1.94, -2.07)	2423.32	2992.5	4566.6
5	Initial: (30, 20) Final: (40, -30)	Initial: (3.75, -2.4) Final: (0.4, -2.0)	1557.12	19792.2	5812.5

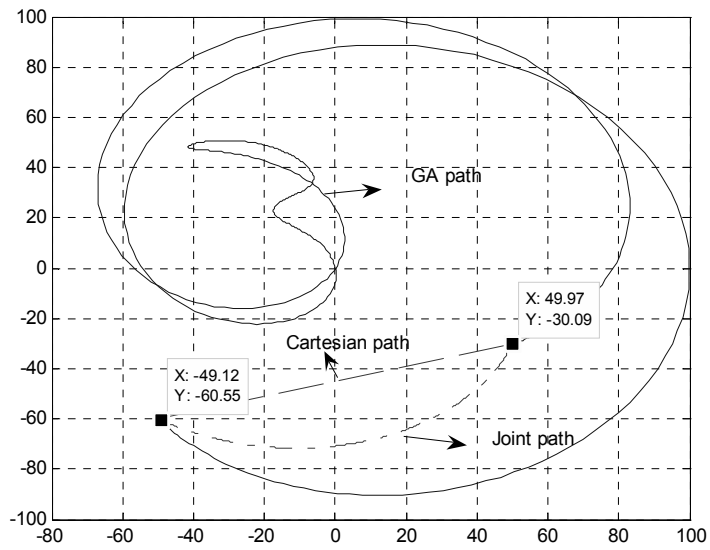
The parameters in bold are the optimal energy paths observed for the two link manipulator. In the observed five simulations, the third simulation has the energy optimal for the Cartesian linear path. The paths for the five cases are shown below in Cartesian space.



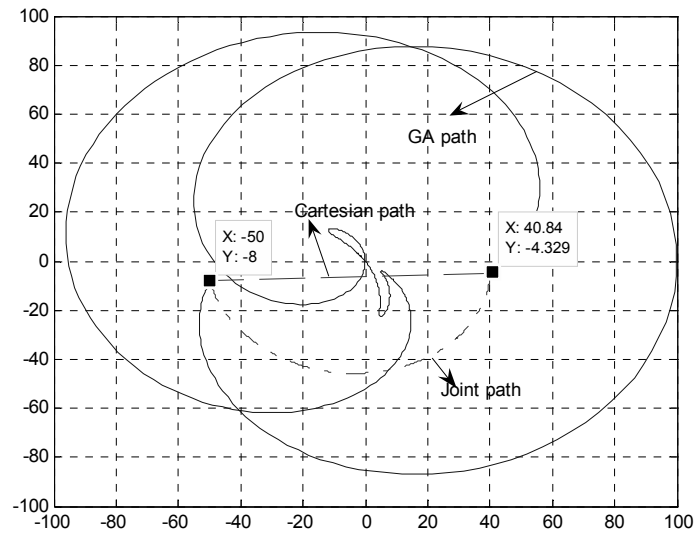
Paths in Cartesian space for case-1



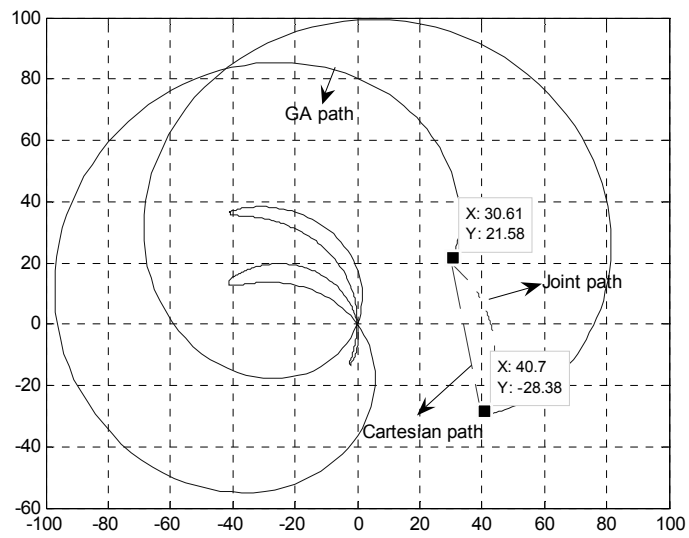
Paths in Cartesian space for case-2



Paths in Cartesian space for case-3



Paths in Cartesian space for case-4



Paths in Cartesian space for case-5

6.3 Matlab/SimMechanics graphics

SimMechanics is a block diagram modeling environment for the engineering design and simulation of rigid body machines and their motions, using the standard Newtonian dynamics of forces and torques. It is a part of the Simulink physical modeling package, encompassing the modeling and design of systems according to basic physical principles. Unlike other Simulink blocks, which represent mathematical operations or operations on signals, physical modeling blocks represent physical components or relationships directly [27].

The system model consists of:

- Body blocks which represents the links L1 and L2.
- Revolute joints for each link.
- Joint actuators for each joint.
- A body sensor at the end-effector to trace the Cartesian path.

The model diagram for the two-link manipulator is developed in SimMechanics as shown in Figure 32. The GA simulations and the trajectory interpolator generate the joint positions in the MATLAB workspace for the joint actuators of the SimMechanics model. When the model is in run mode, the visualization tools of SimMechanics display animations of the manipulator. A screen image of the visualization is shown in Figure 33.

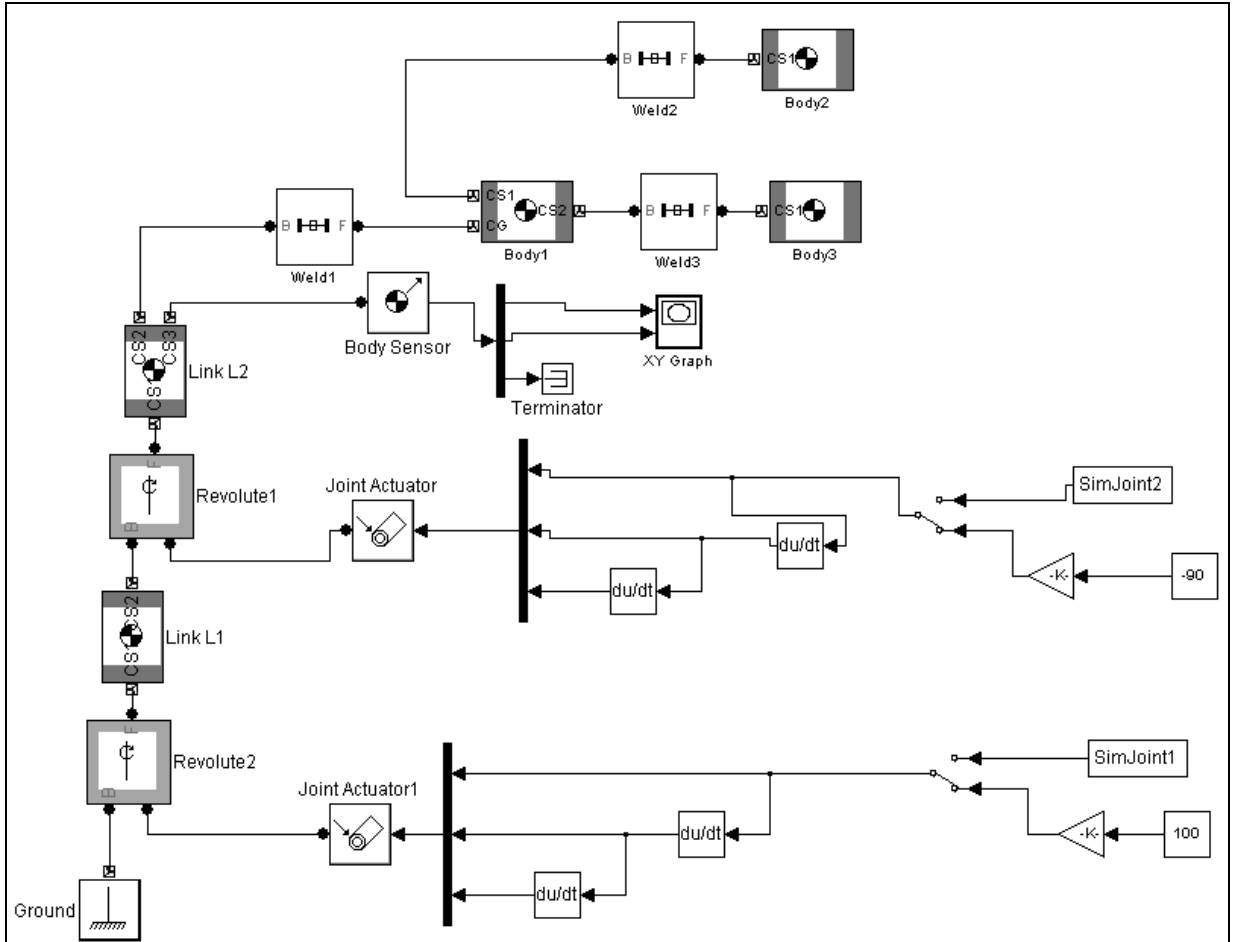


Figure 32: SimMechanics two link manipulator model diagram

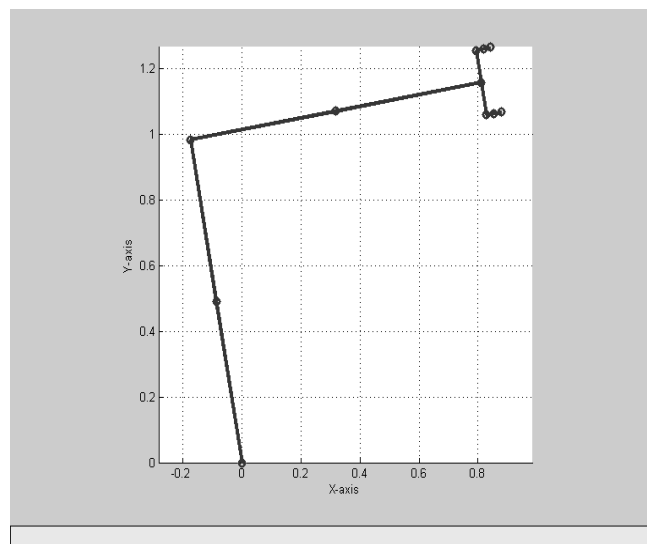


Figure 33: Screen image of the SimMechanics visualization

6.4 Conclusions and future work

A trajectory generation for a two link robot manipulator without the use of the inverse Jacobian matrix was proposed, in which the cubic spline approach was employed. Using simulations, it is verified that the total electrical energy, for given initial and final positions of the manipulator, is approximately minimized by the proposed method. The proposed algorithm is simple and can be easily implemented in real hardware.

The following properties make genetic algorithms for trajectory planning advantageous and attractive:

1. Simple, easy to program and apply.
2. Effective (faster than any random search method).
3. Avoids singularities in the workspace.
4. Approximately optimal energy path can be obtained.

The presented algorithm is a first realization of the evolutionary approach with a GA and leaves open a number of possibilities for further development and improvement. A possible extension for this work would be implementation of a GA approach of trajectory generation for a 6 DOF industrial robot, which is very necessary for practical applications. As the number of degrees or links increases, the inverse kinematics computation would be a challenge. The chromosome length is a function of the degrees of freedom and the number of knots, and hence the GA computation will also be effected.

As with the other evolutionary computation techniques, Particle swarm optimization (PSO) is a population-based search algorithm and is initialized with a

population of random solutions, called particles [25]. Implementation of particle swarm optimization is another approach to solving the optimal trajectory problem for manipulators.

The proposed approach uses the electrical consumed energy as the optimality criterion for the trajectory generation of manipulators. Since minimizing the mechanical energy would almost minimize the electrical energy, it can be used as the objective function in the future work.

In the present algorithm the user has the flexibility of choosing the number of knots in the path, but only at regular intervals of time. A further study would be allowing the user the ability to specify the intermediate knots at different instants of time in the trajectory path.

In this study, we consider robotic manipulators with the mechanical structure- link lengths fixed during the optimization. An evolutionary approach to the robotic structure is presented in [9]. In this line of thought, future work will take into account two GA's. One GA is used to calculate the robot's structure. The second GA would be used to calculate the optimal energy path for the corresponding robotic structure.

Since GNU/LINUX has an advantage of its underlying code being publicly distributed, which is a very significant issue for academic research; it can be used to implement the whole control system for the manipulator [26]. In absence of real robots, CAD models designed with similar mechanical characteristics to those of a real robot could be used for simulation studies.

Motion planning can be applied in static or dynamic environments. In static environments, the objects are static and only the robot moves, whereas in dynamic environments, both robot and obstacles move [28]. So at a future a time, an optimal trajectory path for a manipulator among stationary and moving objects can be proposed.

Manipulators have to operate safely in cluttered, 3-D environments. In order to achieve this, the manipulator has to plan a path through its workspace to its final destination, while avoiding obstacles [12]. Genetic algorithms can be applied for the optimization problem with goals and constraints consisting of planning the end-effector trajectory, avoiding collisions between the robot and the obstacles, and also consuming minimum energy.

REFERENCES

1. John J. Craig , “Introduction To Robotics - Mechanics and Control” 3rd edition, Pearson Prentice Hall, 2005
2. D. E. Whitney, “Resolved motion rate control of manipulators and human prostheses,” IEEE Transactions on systems, Man and Cybernetics, Vol. MMS-10, pp. 47-53, 1969
3. J. Y. S. Luh, Y. L. Gu, “Industrial Robots with seven Joints,” Proc. IEEE International Conference on Robotics and Automation, Mar. 1985
4. J. E .Borrow, “Optimal Robot path Planning Using the Minimum-Time Criterion,” IEEE Journal of Robotics and Automation, Vol. 4, pp. 443-450, August 1988
5. K. Sakamanto, A. Kawamure, “Trajectory Generation for Redundant Robot Manipulator by Variational Approach,” Advance motion control-San Francisco, pp. 378-385, March 1994
6. Y. Davidor, “Genetic Algorithms and Robotics, a Heuristic Strategy for Optimization,” World Scientific, 1991.
7. W. M. Yun, Y. G. Xi, “Optimum motion planning in joint space for robots using genetic algorithm,” IEEE on Robotics and Automation Systems, Vol. 18, pp. 373-393, 1996.
8. A. Rana, A. Zalzal, “An Evolutionary Planner for Near Time-Optimal Collision-Free Motion of Multi-Arm Robotic Manipulators”, Int. Conf. Control, Vol. 1, pp. 29-35, 1996.
9. E. J. Solteiro Pires, J. A Tenreiro Machado, P.B. de Mour Oliveira, “An Evolutionary Approach to Robot Structure and Trajectory Optimization,” ICAR’01-10th International Conference on Advanced Robotics, Budapest, Hungary, pp. 333-338, 22-25/Aug/2001.
10. A. A. Ata, R. T. Myo, “Optimal Point-to-Point Trajectory Tracking of Redundant Manipulators using Generalized Pattern Search,” International Journal of Advanced Robotic Systems, Volume 2, Number 3, pp. 239-244, 2005.
11. S. S. Roy, “A Genetic-Fuzzy Approach for Optimal Path-planning of a Robotic Manipulator among Static Obstacles,” The Institution of Engineers (India), Volume 84, pp. 15-22, May 2003, Available online, <http://www.ieindia.org/publish/cp/0503/may03cp4.pdf>.
12. L. Blackmore, B. Williams, “Optimal Manipulator Path Planning with Obstacles using Disjunctive Programming,” American Control Conference, 2006, Available online, http://mers.csail.mit.edu/papers/BlackmoreWilliams_ACC06.pdf.

13. W. J. Book, "Recursive Lagrangian dynamics of flexible manipulator arms," *International Journal of Robotics Research*, pp. 87-101, 1984
14. K. Chang, O. Khatib, "Manipulator Control at Kinematic Singularities: A Dynamically Consistent Strategy," *Proc. IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, Pittsburgh, Vol. 3, pp. 84-88, August 1995.
15. C. DeBoor, "A Practical Guide to Splines," Springer-Verlag, New York, 1978
16. K. Atkinson, "An Introduction to Numerical Analysis," Wiley, New York, 1989.
17. R. P. Paul, H. Zong, "Robot Motion Trajectory Specification and Generation," 2nd International Symposium on Robotics Research, Kyoto, Japan, August 1984.
18. R. Taylor, "Planning and Execution of Straight Line Manipulator Trajectories," in *Robot Motion*, Brady et al., Editors, MIT Press, Cambridge, MA 1983.
19. K. S. Fu, R.C. Gonzalez, C. S. G. Lee, *Robotics Tutorials*, IEEE Press, 1990.
20. J. Y. S. Luh, M. W. Walker, R. Paul, "On-Line Computational Scheme for Mechanical Manipulators," *Transactions of the ASME Journal of Dynamic Systems, Measurement, and Control*, pp. 69-76, 1980.
21. A. R. Hirasawa, A. Kawamura, "Proposal of Trajectory Generation for Redundant Manipulators Using Variational Approach Applied to Minimization of consumed Electrical Energy," *Advanced Motion Control, AMC'96-MIE*, IEEE Proceedings, pp. 687-692, 1996.
22. D.B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Transactions on Neural Networks*, Volume 5, Issue 1, pp. 3-14, Jan 1994.
23. K. Watanabe, M. M. A. Hashem "Evolutionary Computations -New Algorithms and their Applications to Evolutionary Robots," *Series: Studies in Fuzziness and Soft Computing*, Vol. 147, Springer, 2004.
24. M. Jamshidi, "Robust Control Systems with Genetic Algorithms," Reading, CRC Press, 2002.
25. Y. Shi, "Particle Swarm Optimization," *IEEE Connections*, 2004.
26. D. Aarno, "Evaluation of Real-Time Linux Derivatives for Use in Robotic Control," Master's thesis, Royal Institute of Technology, Stockholm, Sweden, Aug, 2004, www.nada.kth.se/~bishop/resources/rtos.pdf
27. SimMechanics, Users Guide, The Math Works Inc., June 2004.

28. P. Fiorini, Z. Shiller, "Time Optimal Trajectory Planning in Dynamic Environments" *Journal of Applied Mathematics and Computer Science*, special issue on Recent Development in Robotics, Vol. 7, No. 2, pp. 101-126, June 1997.