Thesis Report

on

Helical Interpolation for Welding using Articulated Robotic Arm

Submitted in partial fulfilment of the requirement

for the award of degree of

Master of Engineering

In

CAD / CAM & Robotics.

Submitted By

Rameshwar Cambow

801081025

Under the Guidance of

Mr A.S Jawanda

Associate Professor, MED

Thapar University, Patiala.

Department of Mechanical Engineering,

Thapar University, Patiala. – 147004.

July 2012

CERTIFICATE

This is to certify that the work in this thesis report entitled **"HELICAL INTERPOLATION FOR WELDING USING ARTICULATED ROBOTIC ARM"** submitted in partial fulfilment of requirement for the award of **Master of Engineering Degree in CAD/CAM & Robotics** in Mechanical Engineering Department of Thapar University, Patiala, is an authentic record of work carried out by me under the guidance of **Mr A.S. Jawanda, Associate Professor, Mechanical Engineering Department, Thapar University, Patiala.**

The matter embodied in this report has not been submitted in part or full to any university or institute for the award of any degree. The matter in this report is an original compilation by me with referred material from published literature duly acknowledged.

Dated: 14th Aug, 12.

Rameshwar.

Rameshwar Cambow

801081025

M.E. (CCR)

This is to certify that above declaration made by the student concerned is correct to the best of my knowledge and belief.

14/8/12

Mr A.S JAWANDA

Associate Professor, MED

THAPAR UNIVERSITY, PATIALA

Countersigned by:

3/8/12

DR. AJAY BATISH

PROFESSOR & HEAD-MED

THAPAR UNIVERSITY, PATIALA

DR. S.K. MOHAPATRA

DEAN, ACADEMIC AFFAIRS

THAPAR UNIVERSITY,

PATIALA

ACKNOWLEDGEMENT

Words are often less to reveal ones deep regards. With an understanding that work like this can never be the outcome of a single person. I take this opportunity to express my profound sense of gratitude and respect to all those who directly or indirectly helped me through the duration of this work.

I take the opportunity to express my heartfelt adulation and gratitude to my supervisor Mr. A.S Jawanda for his unreserved guidance, constructive suggestions, thought provoking discussions and unabashed inspiration in the nurturing work. It has been a benediction for me to spend many opportune moments under the guidance of the perfectionist at the acme of professionalism. The present work is testimony to his activity, inspiration and ardent personal interest, taken by him during the course of his work in its present form. I am grateful to Dr. Ajay Batish, Professor and Head, Mechanical Engineering Department for providing the facilities for the completion of the work and always being a shelter in the odd days.

I take pride of myself being son of ideal parents and a brother of caring sister for their everlasting desire, sacrifice, affectionate blessings, and help, without which it would not have been possible for me to complete my studies.

No words acknowledge the support I received from my friends for their valorous help and co-operation. I would like to thank all the members and employees of Mechanical Engineering Department, Thapar University, Patiala for their everlasting support. Above all, I express my indebtedness to the "ALMIGHTY" for all his blessings and kindness.

Rameshwar Cambow

801081025

ABSTRACT

Robots are flexible, reprogrammable automated machines which are constrained by their physical configuration to perform tasks which are easily programmable considering degrees of freedom of their joints. Robotic hardware that has been used for particular applications can be utilized for new tasks which its structural configuration is not designed by using a customised path planning. This thesis addresses these problems with the RHINO educational robot original software interface that limited the RHINO to tasks suited for a jointed arm configuration only. A new software interface, based on a popular high-level structured language, is introduced which will make the RHINO robot useful for paths which are traditionally not used. Specifically, the inverse kinematics problem is solved for the robot at hand and used in an experiment to calculate on-line the needed position and orientation of the end effector. This important capability was not possible with the commercial original interface of the RHINO. Other advantages of the newly introduced interface; software library units; open the way for more educational benefits of the RHINO in Robotics, Digital Control, Engineering projects, and research.

In present scenario, there is a need to have path interpolation for articulated robotic arm so as to follow user defined paths to be useful for inspection, welding & machining operations. This robot is used after adapting its controller to be programmable so that it can generate steps for motors employed at the joints & these motors will further control the joint angles. For more understanding & implementation of the idea, a test case of welding a cylinder liner of an IC engine is taken into account, this process takes place to recondition the worn out engine and it is done by filling material on the inner diameter of cylinder liner. To perform the described task, a consumable electrode used for arc welding is held by end effector which will follow helical path by considering the rate of electrode consumption also. To follow this path by the tool tip, inverse kinematics program is used and output of which is verified by graphical comparison of the achieved & desired path of the electrode. This is further verified & visualised using a Robotic Simulator built in standard mechanism design CAD/CAE package of mechanism design extension in Creo software.

TABLE OF CONTENTS

1. INTRODUCTION

Robots are widely used in number of industries to raise the production without compromising with accuracy. Mostly robots are preferred to perform repetitive tasks with high precision which is not possible by humans. Now a day, robots are used for several applications in manufacturing and manipulation tasks. At present, most of the robots are employed to perform dedicated task with almost negligible amount of flexibility. Therefore, for every new process, a new robot is required which involves huge amount of investment as the robots are very expensive. Different types of configurations of robots are available which is to be selected according to the task to be performed. The various configurations of the robot are:-

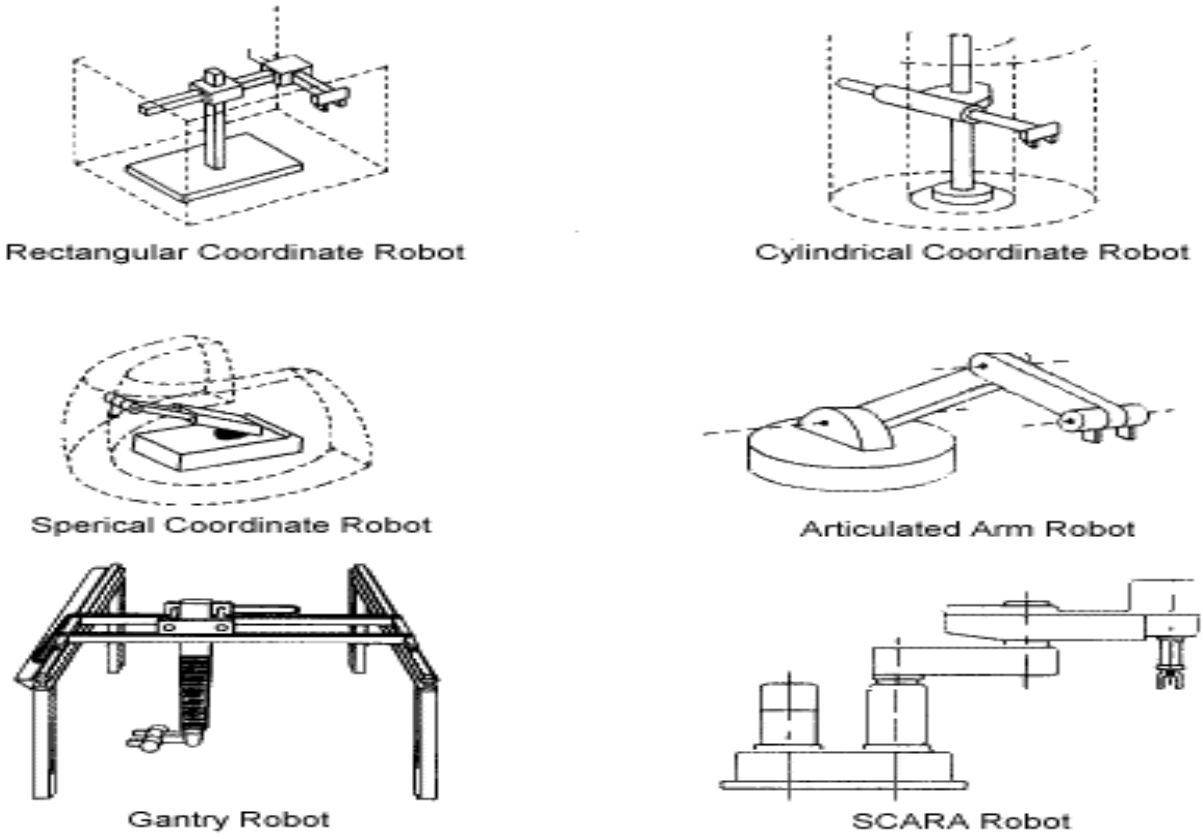


Figure 1.1: Various Configurations of Robot [13]

1.1 Configurations of Robots

Rectangular - These are also called rectilinear or gantry robots. Cartesian robots have three linear joints that use the Cartesian coordinate system (X, Y, and Z). They also may have an attached wrist to allow for rotational movement. The three prismatic joints deliver a linear motion along the axis.

Cylindrical - The robot has at least one rotary joint at the base and at least one prismatic joint to connect the links. The rotary joint uses a rotational motion along the joint axis, while the prismatic joint moves in a linear motion. Cylindrical robots operate within a cylindrical-shaped work envelope.

Spherical or Polar - Also called spherical robots, in this configuration the arm is connected to the base with a twisting joint and a combination of two rotary joints and one linear joint. The axes form a polar coordinate system and create a spherical-shaped work envelope.

Articulated - This robot design features rotary joints and can range from simple two joint structures to 10 or more joints. The arm is connected to the base with a twisting joint. The links in the arm are connected by rotary joints. Each joint is called an axis and provides an additional degree of freedom, or range of motion. Industrial robots commonly have four or six axes.

Gantry Robot - A gantry robot consists of a manipulator mounted onto an overhead system that allows movement across a horizontal plane. Gantry robots are also called Cartesian or linear robots. They are usually large systems that perform pick and place applications, but they can also be used in welding and other applications. Its systems provide the advantage of large work areas and better positioning accuracy. Position accuracy is the ability of the robot to place a part correctly. Gantry robots are easier to program, with respect to motion, because they work with an X, Y, Z coordinate system. Another advantage is that they are less limited by floor space constraints.

SCARA - Commonly used in assembly applications, this selectively compliant arm for robotic assembly is primarily cylindrical in design. It features two parallel joints that provide compliance in one selected plane.

The typical use of all the configurations is limited. Traditionally, to trace a linear path, articulated arm is not preferred. Therefore in an industry possessing articulated arm robot, if there is requirement to follow linear path in any process then conventionally the given task is to be performed using robot having prismatic joint i.e. another configuration of robot is required which is not economically feasible. That is why, there is need to develop solution to the given problem so that flexibility and versatility can be provided i.e. a robot is required which can adapt itself to perform various types of jobs following different paths.

For this given task, literature review is done to find out how much flexibility is imparted and how more flexible robots can be made.

1.2 Test Case: The XR-III Rhino Robot

The work is implemented on XR-III series rhino [12] robot arm made by RHINO ROBOTS, LTD., OH, USA.

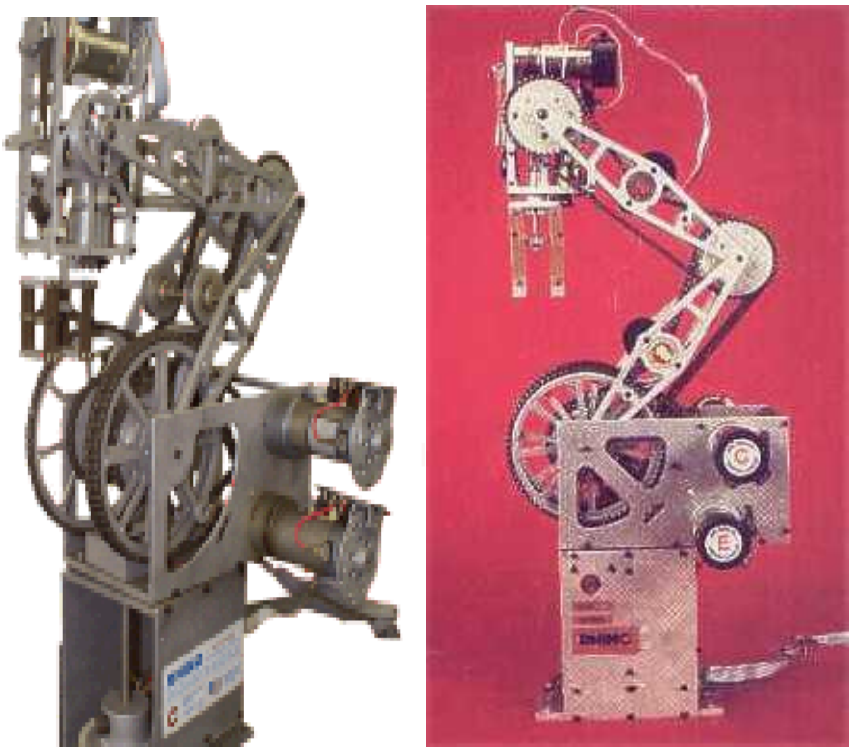


Figure 1.2: Rhino robot [12]

(Five axis, robotic arm with electrical gripper)

The robot is a 5 DOF articulated arm robotic arm. It is an educational robot so has lower payload and accuracy as compared to an industrial robot. As the robot is an old, flood damaged, one it can be used only as a base for making an accurate CAD based simulator.

BASIC SPECIFICATIONS FOR THE RHINO ROBOT ARM

- Vertical Reach: 68cm
- Reach: 60cm
- Lifting Capability (arm extended): 0.45Kg
- Weight of Rhino: 7.7Kg

- Weight of Controller: 12.3Kg

For more details of the Rhino robot arm, refer to Appendix.

1.3 Need analysis

Auto industry is very large consumer of robots. In this era of competition, with the continuous and very frequent change in auto technology, there is the need arising to provide flexibility in robots at shop floor so as they can adapt themselves according to the frequent modifications.

Introduction to Cylinder Liner

To validate the idea presented in this thesis, an example of welding a cylinder sleeve is taken. With the help of this experiment, whole concept can be easily understood and validated also. Starting with explanation about cylinder sleeve or liner. A cylinder liner is a cylindrical part to be fitted into an engine block to form a cylinder. It is one of the most important functional parts to make up the interior of an engine. This is called Cylinder liner in Japan, but some countries (or companies) call this Cylinder sleeve.

Figure 1.3: Different Cylinder Sleeves [28]

Cylinder walls can become very worn or damaged from use. In such cases the use of a sleeve or liner can restore proper clearances to an engine. Sleeves are made out of iron alloys and are very reliable. A sleeve is installed by a machinist at a machine shop. The engine block is mounted on a precision boring machine where the cylinder is then bored to a size much larger than normal and a new cast-iron sleeve can be inserted. The sleeves can be pressed into place, or they can be held in by an interference fit. The interference fit is done by boring the cylinder (between 3 to 6 thousandths of an inch) smaller than the sleeve being installed, then heating the engine block and while hot, the cold sleeve can be inserted easily. When the engine block cools down it shrink fits around the sleeve holding it into place. Once a sleeve has been installed the cylinder needs to be finish bored and honed to match the piston. In this case, material is filled while welding the sleeve then cylinder is bored to required size. The welding operation of the sleeve is shown in the Figure 1.4.

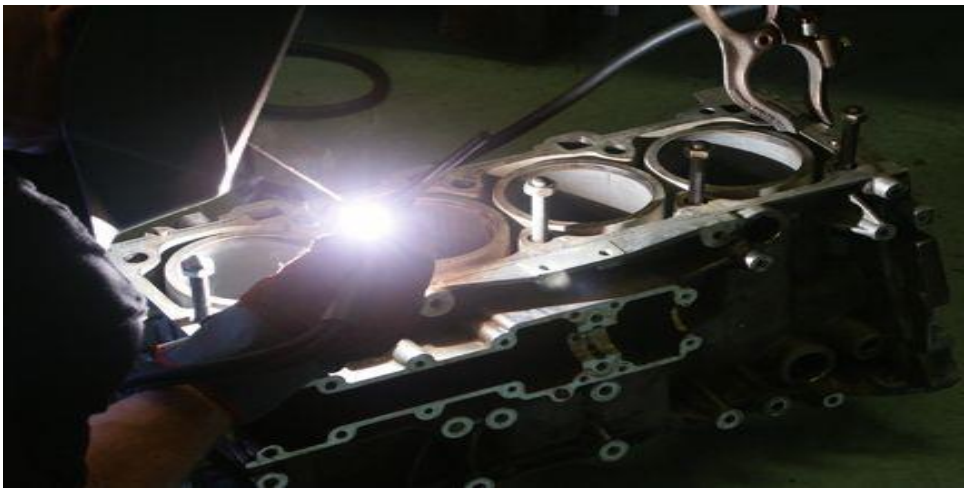


Figure 1.4: Welding inside cylinder to fill the required material [29]

Multi cylinder heavy engines are used in railways, to recondition these engine blocks, there is need to fill the material along the circular shape of the block which is a cumbersome task to be perform manually. Because if the given task is to be carried out by human in case of 12 cylinder engine, it results in over fatigue of labour and poor quality weld which becomes a challenge for quality control. Therefore, the solution to the current problem is the use of robotic arm. To carry out this task, robots with prismatic and cylindrical joints are the most

obvious choice but in case of industry already having articulated arm robot, the given task is conventionally not recommended. So, instead of investing huge amount on new compatible configured robot, the available robot must be adaptable to perform the desired task efficiently.

Requirements for the proposed implementation of Helical Interpolation using the Rhino Robotic Arm

1. The controller hardware or software provided with the rhino robot was not capable of Helical interpolation So the controller had to be customised to an open architecture controller, which is able to implement user generated complex paths.
2. Software for generating complex paths is required.
3. There is no simulator to test the generated paths.
4. The structure of the robot in the lab is very old and effected in floods of 1993 so has to be repaired and the accuracy of chain drives is adversely effected. So the physical verification has to consider the large errors which come in backlash, slippage, and loose fitting of parts of the drives. Thus a simulated path verification has to be implemented. The simulator has to be designed for the required accuracy.

2. LITERATURE REVIEW

After analysing the need of providing flexibility in robots, the survey of various suitable controllers is done so as the need can be incorporated into the controller.

Some surveyed controllers are: -

- Orangutan Controllers
- ‘arbotriX’ Controllers
- Arduino Controllers
- Peripheral Interface Controllers (PIC)

Orangutan Controller [24]: Pololu’s Orangutan robot controllers are combinations of microcontrollers and additional hardware useful for controlling robots. All Orangutans feature Atmel AVR microcontrollers, multiple H-bridges for direct control of DC motors, and most units have integrated LCD displays. The Orangutan is small enough for integrating into a small robot, rather than being the small robot.

‘arbotriX’ Controller[25]: The ‘arbotiX’ robocontroller is an advanced control solution for small-to-medium size robots. It incorporates a powerful AVR microcontroller, XBEE wireless radio, dual motor drivers, and 3-pin servo-style headers for input output. The arbotiX robocontroller is specifically designed to control robots built using Bioloid servos, although it is also a very high-end solution for powerful rovers. It is intended as a high level development tool for building more sophisticated robots.

Arduino [26]: Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments. Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the

Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be standalone or they can communicate with software running on a computer (e.g. Flash, Processing, MaxMSP).

Peripheral Interface Controllers (PIC) [19]: PIC is a family of modified Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1650 originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to "Peripheral Interface Controller". PICs are popular with both industrial developers and hobbyists alike due to their low cost, wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, and serial programming (and re-programming with flash memory) capability.

Introduction to development of Computer Controlled Devices [1]

A common way to modularize the development of computer controlled devices is to focus on three problems:

a) The definition of an interpretive language capable of describing the tasks to be done.

b) The construction of an interpreter/mechanism which accepts such descriptions and executes the task.

c) The creation of suitable applications programs for facilitating the generation of the description of specific tasks in terms of the interpretive language.

Frequently the interpretive language is designed with a degree of device independence so that the work invested in its development can be amortized over many products. For example, various commercially available pen plotters and laser printers fit this pattern and a number of motion control products are also of this type. The main problem in motion control is to simultaneously achieve adequate speed and expressiveness.

From the point of view of robotics, perhaps the most innovative features are the introduction, through

i) Of programmable compliance at the lowest level and the use of a combination of table lookup and matrix multiplication, using the idea of exponentially separable functions,

to implement the sweeping generalizations of the inverse kinematics problem embodied

ii). Analysis is given supporting the expressiveness and computational tractability of a solution based on these choices.

2.1 Introduction to Robot Controller

Motion control is now recognized as a key technology in mechatronics. The robustness of motion control will be represented as a function of stiffness and a basis for practical realization. Target of motion is parameterized by control stiffness which could be variable according to the task reference. However, the system robustness of motion always requires very high stiffness in the controller. In the 1970's, industries began to replace mechanical elements with electronic ones to achieve higher reliability and less maintenance. Also the mechatronic devices were designed to occupy smaller space in the final products. Totally function of reliability, availability, and serviceability has been very much improved in relatively more compact products. In the 1980's, a remarkable progress in mini- and microcomputers and power electronics technology made it possible to improve the performance of motion. For example, vector controlled induction motor has higher cut-off frequency almost up to three times in the speed control loop compared to the same-sized dc motor. Following these results, the novel theories of control were tested in such mechatronic systems. In the late 1980's and the early 1990's, mechatronics seemed a showcase of various applications of control theories. The phenomena observed in the early 1990's also came from the so-called "software-servo technology." Generally major part of software applied to motion control carries out the indispensable routines for diagnostics and sequential procedures. Only small area is assigned for programming control algorithms. The area was hardly sufficient for conventional PID controller. Recently the fast processor has gradually enabled more complicated algorithms within a shorter sampling time. Since the software-servo technology has generated more room for control algorithms, higher performance and flexibility have been realized without additional investment. Then the novel algorithms have gained high evaluation from the practical viewpoint because the quality of motion was improved. The motion control is now recognized as an important area in mechatronics. The physical meaning is emphasized rather than mathematical exactness. As is well known, control and estimation are twin aspects of system design. The fact holds in motion control. The robust control and the estimation of parameters have the same basis. The several

examples shown later seem different approaches; however, the single interpretation is possible from the physical viewpoint. Firstly [4], stiffness is defined in relation to various motion controls. This concept leads to both the meaning of robustness and the general structure of motion control. Then it points out the necessity of modification against flexible structure, a mechanical system governed by the Lagrange equation is represented both geometrically and dynamically. The kinematics is represented as a set of algebraic equations which gives constraints of motion. The dynamics is a set of differential equations based on dynamic equilibrium of force. A motion controller generates a set of inputs to the actuators according to motion reference. A motion reference is synthesized in the reference generator. The sensor signal, the database and the commands from other motion systems and/or human operators are input signals for the reference generator. There will be some intelligent process with composite structure in the reference generator. The general motion control totally consists of the motion controller and the reference generator.

From the control point of view, the output of the motion will be position and/or force. A simple case is continuous path tracking; however, the need for force control is increasing because the industrial demand to the dexterous motion is growing up. Iterative solutions for finding the inverse kinematics solution of industrial robots have been proposed by Uicker et al. Recently, Nearchou [2] has proposed an evolutionary approach based on a modified binary coded genetic algorithm (GA) to obtain a unique solution for the inverse kinematics problem of non-redundant industrial robots and redundant robots. The issue of multiplicity resolution of an industrial PUMA robot has been solved through the minimization of total joint displacement and the closest solution in the joint space relative to the current configuration is evaluated. The superiority of the evolutionary approach over the well-known pseudo-inverse method and the simple binary-coded genetic algorithm is established in this work.

Moreover, the evolutionary approach does not require the computation of the Jacobian matrix so that any problem related to the inversion of this matrix, like singularities, is overcome. However, the proposed approach suffers from certain limitations. A two level binary-coded GA [3] is proposed wherein potential solutions are evaluated by the high level GA and incremental changes are evaluated by the low level GA until a global optimum is

achieved. This is equivalent to the use of a penalty parameter for satisfying the robot kinematics equations. Although genetic algorithms do not use any gradient information, they may not be free from the distortion effect caused due to the addition of the penalty term with the fitness function.

Moreover, the multiple configurations of an industrial robot existing due to the multimodal nature of the inverse kinematics problem are not available at the end of the search since only the best solution based on the minimization of total joint displacement is evaluated.

The main functions of a robot controller are:

- User interface,
- Data storage,
- Motion planning,
- Real-time control of joints' motion,
- Sensor data acquisition,
- Interaction and synchronization with other machines.
- Interaction with other computational resources.

2.2 Welding With Robot

Remote Laser Welding [3]

Another case is presented as task oriented programming system for remote-laser-welding (RLW) with conventional optics, which considers real work piece data and process limits for the calculation of robot paths. This approach relies on an augmented-reality-based user interface for effective 3D-interaction and task definition. A powerful task and motion planning system has been developed, that transfers task descriptions to optimized, executable robot operations. The resulting system offers fast and efficient spatial interaction methods that allow even novel users to quickly define operations on a task-level. The approach simplifies and accelerates the programming process, consequently leading to a reduction of cycle time for a RLW-task by more than 30%.

The maximum focal length of a focused laser beam for a given spot diameter depends on the beam parameter product (BPP) of a laser source. For a long time, the high BPP of conventional high power lasers like the Nd:YAG solid-state laser, limited the welding distances between optics and work piece to about 250 mm. Recent laser developments such as the fiber laser, improved the BPPs while operating distances were increased to one or more meters. Joining with long focal lengths is commonly termed remote-laser-welding (RLW). The main advantage of RLW is an increased positioning speed of the focus spot on the work piece by small deflections of the laser beam, leading to a significant reduction of cycle time. For beam deflection, several approaches are subject of research, e.g. galvanometer driven mirrors in scanner units. An alternative is the use of an industrial robot with a simple long-range optic. Refer Figure 2.1.

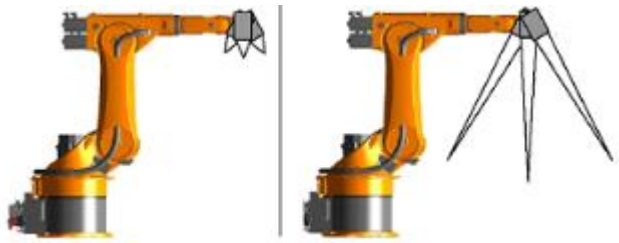


Figure 2.1: **Conventional laser welding (left) and remote-laser-welding with conventional long-range optic (right). [3]**

A central challenge for a broad usage of RLW is programming of an optimal robot movement with respect to cycle time. The quality of welds, expressed in welding depth and width can be manipulated by adjustable process parameters as laser power or welding speed. These parameters must be aligned, e.g. to material properties or plate thickness.

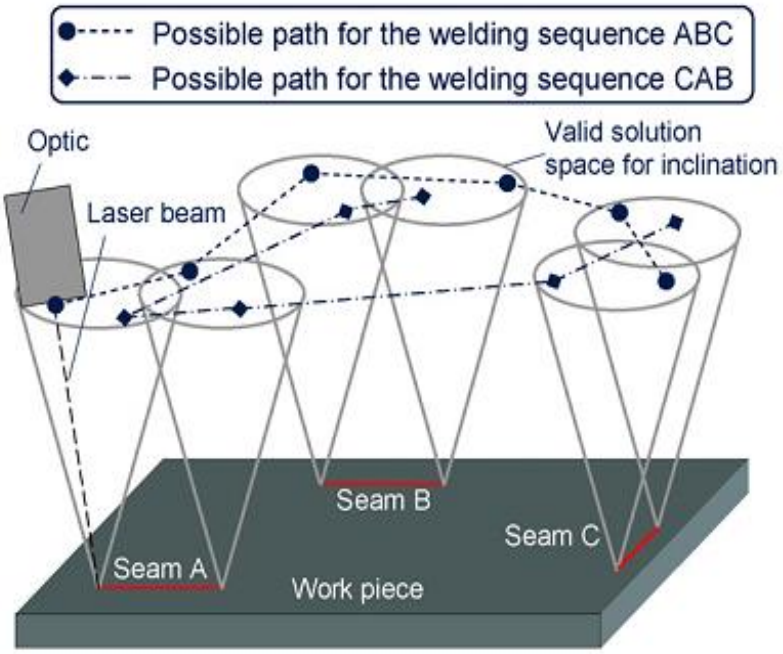


Figure 2.2: Infinite number of possible optic paths within valid inclination angles. [3]

Other parameters as sequence of welds or inclination angles of the laser beam have no or just minimal effect on the welding quality, but a large influence on cycle time. Within this range, an infinite number of possible robot poses for positioning and welding exist. For finding a good solution for a given welding problem, a powerful programming and optimization system is needed.

2.3 Programming Welding Robot

Current programming approaches [14]

Today's industrial relevant programming methods can be roughly classified into online and offline techniques. Online approaches like Teach-In, describe robot paths by guiding the real robot manually to discrete positions. Intelligent sensor aides for programming can assure that the robot is still on the right path but cannot create optimal paths from scratch. Offline methods mostly use 3D-simulation systems with CAD-functionality for definition of the robot setup and thereby transfer the creation of discrete positions from the real to a virtual world. Online programming is a simple but rather slow programming technique. Especially in the case of RLW, the tool-center point (TCP) is far from the tool flange, such that guiding the

robot manually is difficult and time-consuming. To try other inclination angles, manual manipulation of these positions must take place, combined with additional effort for tests and determination of cycle time improvement. Offline simulation systems are powerful tools for modelling and planning of robot systems. In addition, numerous path planning algorithms for offline simulations exist, mostly addressing collision avoidance. However, for RLW a dedicated support for optimizing welding sequences and robot motions is lacking.

Also, as divergences between the real work pieces and the CAD-model can rarely be completely avoided, offline generated robot programs must often be adapted with time consuming online methods. Therefore, an efficient and intuitive support for defining the welding process combing model data and the real work pieces are required.

Programming approach

A RLW task is complex and contains variable and constant attributes. In order to optimize a path within the variant attributes, proposes a task-level programming system, composed of two elements, as it is clear from the Figure 2.3.

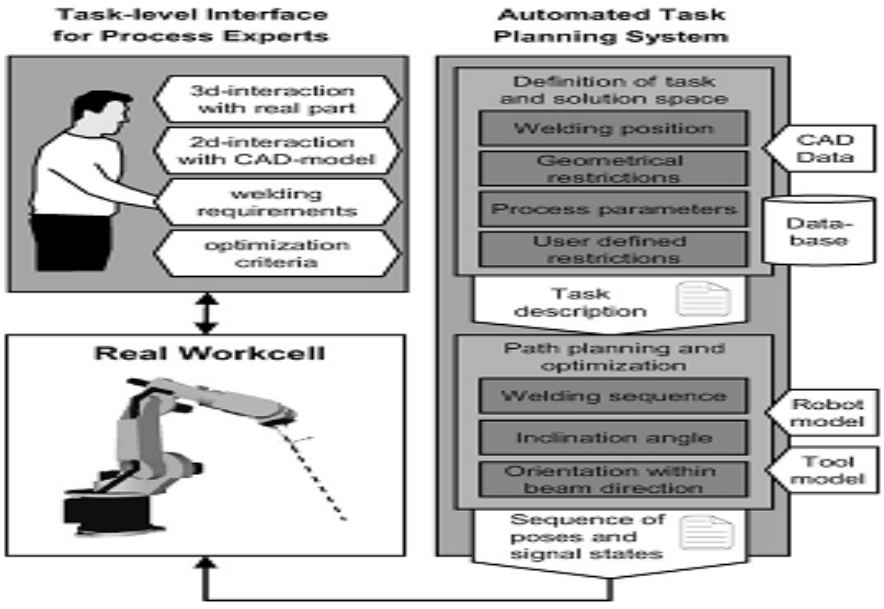


Figure 2.3: Concept of a task-level programming system. [14]

Path planning and optimization

For path planning and optimization, computer-based calculation algorithms have been developed, which use the task description as input data. Because of the complexity of the optimization, the problem is divided in several sub-problems. The two central ones are discussed.

Welding sequence

The first step of optimization is the calculation of an adequate welding sequence. It is assumed, that the shortest Cartesian path between all seams results in the shortest cycle time for a welding problem. Therefore the calculation can be referred to the solution of the ‘‘Travelling Salesman Problem’’ (TSP). For solving the TSP, a simple heuristic approach has been developed, consisting of five steps in Figure 2.3. The first is the determination of an initial, randomly chosen sequence, in which every seam is reduced to one point. The second one is the iterative swapping of two random points and the random insertion of a single point within the sequence. The next step determines a random initial welding direction for every seam which is optimized by trying a change of direction within step four. Finally the longest overall distance defines the start and endpoint of the sequence. This way a short Cartesian path between all seams can be determined.

Variation of inclination angles

It is assumed that a smooth paths of the optic results in improved robot movements regarding to cycle time, because of avoiding main axis oscillation in Figure 2.2. The optimization starts with an initial optics path as in Figure 2.2, described by starting inclination angles. These angles are calculated in a function, regarding reachability of the robot. Mostly the initial path is chiseled. To smooth the path, it is defined, that every point Pn has a mass, affecting the points Pn1 and Pn+1 by a force. The force grows quadratic ally with the distance between the points. It causes a small displacement of Pn as shown in Figure 2.2. An iterative calculation, which randomly displaces every point except the first and the last one, causes a smoothened optics path. The found solution is finally transferred to robot axis positions for every interpolation cycle. Also signal information as laser on/off or laser power settings are added.

Welding Automation in Ships [9]

Over the last few decades, there have been a large number of attempts to automate welding in the shipbuilding process. However, there are still many non-automated welding operations in the double-hulled blocks, even though it presents an extremely hazardous environment for the workers. And, the hazards come about mainly because of the dimensional constraints of the access-hole. Thus, much effort has been recently directed toward the research on compact design of the fully-autonomous robot working inside of the double-hulled structures.

However, the manufacture of double-hulled ships is more time consuming and expensive than that of single-hulled vessels. Here the manufacturing process used to obtain the closed block that is a sub-module of the double-hulled ship wall. A bottom shell and an open block are assembled separately using welding processes where the bottom shell is composed of a wide steel plate with several reinforcing longitudinal stiffeners welded to it in parallel. Forming the closed block is more complicated; since it is an enclosed structure, the temperature gets hot and is in the range 40–50 degrees during the summer, and it is often too dark to freely carry out tasks, even during the daytime. However, human workers currently execute this welding process, working inside the enclosed space surrounded by the top shell, the bottom shell, a pair of transverse web floors and the girders [9]. This manual welding process inside the closed block represents one of the most difficult and hazardous tasks to human workers in the shipbuilding industry. Moreover, the welding robot, which is currently used in the open blocks, with a 6-axis articulated manipulator, cannot be used in the double-hulled block as the overhead gantry crane cannot approach the inside of the double-hulled block. This is because the overhead gantry cranes are installed on the ceiling of the shipyard. Therefore, it becomes clear from the current status of the welding process in the double-hulled structures, that there is a great need for an acceptable solution based on a robotic system that can move around within the closed block, to weld the contacting boundary of the top and the bottom shells, with mobile functions or other suitable alternatives.

ROBUSTNESS OF MOTION CONTROL [7]

Concept of Robustness

Various specifications for motion in industry require versatile ability in the controller. An efficient way to overcome this problem is to divide the function of motion control into two parts. Control flexibility is suitably realized in the motion reference generator since a kind of intelligence is indispensable in this part. It is necessary to track the motion reference accurately in the motion control part. The more intelligent a motion reference generator becomes, the more robust a motion controller should be. This is a kind of master-slave structure. There is an interpretation on robustness of motion controller, which makes the conception visible in mind. Suppose the moving body whose position is controlled along the predetermined path. Such a rigid body should knock down or break any obstacles on the path and go forward to the end of path, if a motion controller is ideally robust. So-called obstacle avoidance issue is solved in the motion reference generator by synthesizing an appropriate reference of trajectory. The robustness of the motion controller assures to the utmost the "high-fidelity" to the input reference.

Actuation

The dynamical equation is excited by input force. Most of mechatronic systems adopt electrical actuator for the purpose. In a typical electric drive system. Most of power converters use switching devices for power control. The regulation of torque highly depends on the switching frequency. Since the recent power converter uses Insulated Gate Bipolar Transistor (IGBT's), Field Effect Transistor (FET's), and so on, for fast switching, the current feedback includes high gain inside the feedback loop and the torque current follows the current reference with delay of less than from 50 ps to 1 ms. The torque itself is produced by electromagnetic interference of current and magnetic flux.

There are three types of the interference. The stepping motor is widely used for simple positioning. It has a similar characteristic of synchronous motor, however, it generates high torque ripple and is not appropriate for fine and smooth motion. Each motor generates torque by the product of torque current by field. The field and the torque current is controlled to be orthogonal to each other. Then by integrating all the torque par small piece of surface of rotor, the total generated torque T, is given simply as Kt is a function of flux

position and expanded in Fourier series and is called a torque coefficient. I, is torque current. Fast switching devices make the power converter with feedback of torque current as a virtual current converter.

Disturbance

It is necessary to define the equivalent disturbance in order to consider the robust control of motion actuated by electric motor. The explanation and the interpretation of robustness and stiffness in motion control lead to definition of disturbance. The general definition for single-input and single output (SISO) linear system is discussed.

Position Control

Positioning is one of the important application in motion control. There are two kinds of industry requirements to the positioning:

• PTP path (Point-To-Point Path)

• CP path (Continuous Path)

The trapezoidal profile of speed reference is used for PTP path tracking. During acceleration and deceleration, the period of constant acceleration is controlled to attain maximum speed.

As to CP tracking, a trajectory of motion is predetermined and it is possible to know the several steps ahead at any time. If the motion reference generator knows two steps ahead, the velocity and the acceleration reference are calculated as well as position reference. The robust motion controller makes the motion system an acceleration controller.

Force Control

In the industry, the current control feedback has been widely used as a torque control. This loop has only a function to make a power converter to be a controlled current source as previously mentioned. The target of force control is a control of force at the end-effector accurately. The robustness of the force control system is also required. It is also a basis for force control and there should not be high forward gain to position in order to keep stiffness tc as low as possible. If the force sensor is ideal, there would be no forward gain for the position and zero stiffness is attained. However, very small deviation proportional to the imposed force could exist in the force sensor and the robustness will be suffered.

There are two categories for force control:

• Noncontact motion

• Contact motion.

In noncontact motion, a force control is substantially an acceleration control. An end-effector moves along force reference until it collides with a fixed environment. In contact motion, there will be a force sensor between the end-effector and the control object. The sensor will detect very small deviation proportional to the imposed force. Then at that moment, a mechanical loop including environment is set up, where k, is stiffness of sensor and environment, de is viscosity of environment, and Kf is a forward gain. It shows that the system is oscillatory with natural angular frequency of m, when the damping of environment de is very small.

Once the end-effector touches the environment, a closed loop is completed. Then the system is oscillatory and the end-effector is repulsed from the environment. When the end-effector separates from the environment, the closed loop is eliminated. Again the end-effector is stable and approaches the environment and touches it again. This process repeats over and over. This hunting phenomenon is overcome by adding damping loop. Generally it is difficult to know the stiffness and the damping of the environment.

Impedance Control

The stiffness of the system is modified to have specified mechanical impedance. In this case, position and force signal are used to generate acceleration reference based on the specified impedance. The stiffness corresponding to the virtual spring coefficient, the artificial damping and the equivalent mass realize mechanical impedance. It is noted that if the gain of the position is zero, the impedance control becomes the force control. The zero gain of the force is the same as the position control. It is possible to turn continuously the motion control to both the position control and the force control by adjusting the control gains in impedance control. In other words, the impedance control is the general form of motion control.

Stability Discussion

The following issues are considered to obtain the Vibration suppression controller.

The controller of the motor portion is designed so that the poles of the system do not cancel the zeros by the motor state feedback. The feed forward compensator is designed so that the location of the zeros is not changed. In the vibration controller based on the external force feedback, PD control is applied to the motor position controller and the external force feedback gain is determined so that the above conditions are satisfied. To ensure the effectiveness of the external force feedback, the system stability is analyzed.

Models of Robot Dynamics for Control System Design [7]

The robot is powered by electromechanical drives, consisting of DC or AC motors in series with a harmonic drive used for speed reduction. (A harmonic drive is a compact, high-torque, high-ratio. in-line gear mechanism incorporating a rigid" circular spline," an elliptical "wave generator, "and a non-rigid "flex spline.") The motors and harmonic drives are mounted on or near the base, with drive torques transmitted to the drive points on the arm links via mechanical links, such as bars or chains. The mounting of drive motors in this manner, in contrast with mounting at the joints, can be shown to provide superior dynamic characteristics.

The dynamic response of the integrated system consisting of the drive motor, harmonic drive linkages and arm may be measured experimentally by recording the motor current. motor velocity (tachometer feedback signal), and arm motion (integrated accelerometer signal) as the servo is excited with random or sinusoidal input signals. For a given (small) signal level, the drive system may be characterized by a linearized frequency response function.

The existence of resonant behavior inside the robot motion-control loop has a dramatic impact on control system design, providing the motivation for development of the models described below. For example, one approach to improving robot motion-control performance is to decouple the dynamics of the robot links through nonlinear control. The decoupling action performed by the controller is particularly significant in the same frequency range as the above resonant behavior. Decoupling controllers that do not take this

strong resonance into account have little chance of successful implementation. Similarly, adaptive control is an alternative approach, proposed frequently; however, from adaptive control theory, it is known that "un modeled dynamics," such as this drive system resonance, lead to severe stability and robustness problems. The existence of drive system interactions does not preclude future use of nonlinear or decoupling control strategies; rather, it demands that realistic robot and drive system models be used in development of implementable.

A detailed examination of the behavior of contemporary manipulators reveals a contrasting perspective: the dominant behavior of the arm is represented by decoupled models for each motor and drive unit. In a sense, the articulated has characteristics similar to those of a Cartesian robot.

There are a number of factors that lead to this conclusion:

• The design of the arm, in terms of the mass distribution of its links and motor locations, inherently minimizes cross coupling effects. This conclusion has been derived analytically, and has been verified through simulation by the authors.

• The high-gear ratios employed in most drive units cause the torques resulting from cross-coupling effects as reflected back to the motors to be minimal.

• Realistic limitations on motor torques prevent the arm from reaching velocities of sufficient magnitude for cross-coupling terms to become significant.

• Realistic trajectories for robot arms in actual manufacturing applications rarely require extremes of velocity and acceleration.

Robots exhibiting this "Cartesian-like" behavior are increasing their penetration of the robot market in most manufacturing applications. This suggests that in the future, motion controls designed to compensate for cross coupling effects will have rather limited benefit in terms of realized performance gains.

Nonlinearities in Robot Arm and Drive Systems

Nonlinearities associated with robot arm geometry and dynamic interactions among the arm links have been long recognized as important to the design of motion controls. In addition to

robot arm nonlinearities, nonlinear characteristics of elements in the drive system play an important role in robot control system design.

Principal nonlinearities include:

(a) Stiffening spring characteristic of the harmonic drive

(b) Viscous plus Coulomb friction damping for both the motor and load inertias.

(c) Current limiters in the motor control loops.

MOTION-CONTROL PERFORMANCE REQUIREMENTS [15]

The performance of motion controls is best evaluated in the context of requirements during manufacturing applications. In this section, we describe the results of simulation of the robot dynamics and motion controls for two representative cases.

Simulation of Motion-Control Performance

Realistic simulations of robot dynamic response have been developed and validated experimentally for each of the major components that comprise the combined robot arm and controller system. Principal elements in the simulation include:

• Kinematics and dynamics of the multiple link robot arm, consisting of revolute and prismatic joints.

• Models for the robot drive systems, including current-limited DC motors, harmonic drives, and inertia and flexibility effects in the mechanical elements connecting motors to arm links.

• Digital controls for individual joint servo loops, including finite sampling rate and amplitude quantization effects. The finite sampling rate results from through constraints on the microprocessors used in the servo control, while amplitude quantization results from finite resolution of encoders sensing joint angles.

• Algorithms used for coordinate transformations, path interpolations, and special programming features, such as rough positioning, dwells, and smooth decelerations.

Models for Control System Design [15]

• Drive system flexibility plays a critical role in robot motion-control design. Fourth-order dynamic models exhibiting resonance/anti resonance behavior accurately represent the interaction of the drives with arm links. Wrist drive systems can often be represented with simpler, non-resonant models.

• Modern designs for articulated arms exhibit behavior that is "Cartesian-like," in that the effects of nonlinear cross-coupling terms are minimal. This permits more direct design of motion controls assuming decoupled axes for the robot.

• Torque limits on the motors and drives present the most significant nonlinearities in the motion-control problem.

Performance Requirements [15]

• Most manufacturing applications employ robot end-effector trajectories that do not produce excessive dynamic path errors.

• In most cases, dynamic overshoot errors that do occur in the robot path are reduced to acceptable levels easily through programmed decelerations in the motion program with minor penalty in total cycle time.

• The most significant factor constraining robot performance is motor torque limits. This underscores the importance of development of strategies for optimal path planning within the bounds of motor output.

Industrial robots have been remarkably progressed and applied to various industrial fields. Recently, open architectural industrial robots, whose kinematics and servo control are technically opened, have been developed. Using such a robot we can explore new skillful applications without conventional complicated teaching.

In the manufacturing industry [2] of PET bottle moulds, 3D CAD/CAM systems and NC machine tools are being used generally and widely, and these advanced systems have drastically rationalized the design and manufacturing process of metallic moulds.

Normalized tool vectors are first generated from 3-axis cutter location data (CL data). The CL data with normal vectors called multi axis CL data can be used for not only a desired

trajectory of tool translational motion but also contact directions given to a mould. Then, the impedance model following force control method is proposed for controlling the polishing force composed of contact and kinetic friction forces. Finally, a CAD/CAM-based position/force controller in Cartesian space by referring such multi-axis CL data is proposed for polishing robots with a ball-end abrasive tool. The surface polishing is achieved by controlling both the tool position along the CL data and the polishing force. The CAD/CAM-based position/force controller is applied to an industrial robot with open control architecture. The effectiveness and validity of a mould polishing robot with the CAD/CAM-based position/force controller are demonstrated through actual polishing experiments.

Design of position/force controller [2]

In general, a system can be properly controlled only if sufficient information about the system behavior is available. Irrelevant information incorporated in the feedback could be useless, but it may generate an undesired disturbance. Good performance can be obtained when enough information is available and irrelevant information is eliminated his rather intuitive observation leads us to the definition of the other design guidelines that should be observed in the design of P/F controller.

• The position sub controller, which operates on the basis of information only about the twist of freedom, should only control the twist of freedom, and not affect the twist of constraint.

• The force controller, which operates on the basis of information only about the wrench of constraint, should only control the wrench of constraint, and not affect the twist of freedom.

Paraphrasing, the guidelines state that there should be no interaction between the P/F sub controllers:

• The position controller should not affect the motion of constraints because there is no information about the motion of constraints in its feedback loop.

• The force controller should not affect the motion of freedom because there is no information about the motion of freedom in its feedback loop.

The above guidelines establish the principle of consistency, based on which there should be no interaction between the sub controllers.

The guidelines can be illustrated by the following selected cases:

• When the end-effector is required to exert a predefined force on a constraining surface while moving along a trajectory on the surface, any influence of the position controller on the contact force is not controllable because no information about the twist of freedom is available to the force controller. In principle, it is impossible to control the contact force accurately when there is an influence from the position controller.

• When there are errors in the geometric model of the constraints, it is possible for the end-effector to loose contact with the constraining surface. In this case, a good performance P/F controller should be able to compensate for modelling errors, that is, the force sub controller of the P/F controller should be able to return the end-effector back to the contact with the constraining surface. However, if the force sub controller affects the twist of freedom, the motion of the end-effector will deviate from the normal to the constraining surface, and some motion will appear in the subspace of freedom. This motion cannot be controlled by the force controller because of lack of information about the twist of freedom in the force control loop.

Other ongoing improvements of the robot technology are related to the user- and application interfaces of the robot controller with the purpose to make robot programming, operation and maintenance simpler even though the complexity of the robot systems is continuously increasing. Wizard-like step-by-step concepts based on graphical representations of robot movements and process actions are realized also for the use on teach pendants, the realistic robot simulation interface is implemented on the robot programming level tools for process modelling are developed to simplify robot programming, [4] PLC functionality for logic control and advanced process- and equipment control is integrated into the robot controllers and remote automatic data acquisition of robot production data for optimization, monitoring, preventive maintenance and fault isolation is further developed. The fault isolation can be based on data from the sensors in the servo loops, special supervision sensors, current and voltage levels in the drive system and virtual sensors using the observer concept. This development will certainly proceed and it is expected that dynamic robot models run in real

time will be even more used for fault detection, fault isolation and diagnosis, based on residual generation and system identification methods.

Model-based control [8] has been found to be very important in robotics in order to fulfill the contradictory requirements on performance improvements and cost reductions. The ongoing development is directed towards more complex kinematic- and dynamic models, more complex multiple-input multiple-output (MIMO) control schemes, bigger variations of static and dynamic model parameters, increasing noise and disturbance levels, larger number of low mechanical eigen frequencies and enlarged non-linearity's. Even if a lot of academic research has been made on all of these aspects, a lot of applied research is needed in order to further improve the model-based robust control of industrial robots. Hand in hand with the further development of model based control there is an important development of model based design using virtual prototyping to improve the performance/cost ratio, reduce the development cost and to be able to shorten the product cycles. Very important is then the mechatronic design approach with development teams consisting of both mechanical design and robot control specialists.

Thus, the development [8] of both hardware and software in the PC-area makes a big impact on the robot controller development and efforts are made to use also technology coming from the telecommunication area. The cost of software development and maintenance is continuously increasing and a long software life time is therefore important, which means that, for example, new efficient software development environments and new concepts for scalable system architectures, open interfaces and communication concepts are important drivers of the robot controller development. Even if the fastest development in robotics is found on the controller side, there is also some technology push on the electromechanical side. Examples are more efficient and bigger compact gear boxes, more cost efficient motors and drive systems, cheaper carbon composite material and more advanced tools for mechanical design.

Industrial robot development has for sure not reached its limits and there is still a lot of work to be done to bridge the gap between academic research and industrial development and to intensify academic research in directions that target new applications and new flexible automation concepts. When developing robot control for real industrial use a lot of

unforeseen problems arise and many of these problems need applied research to be solved. It is then very important with a close collaboration between researchers, industrial robot developers, automation system builders and robot users. Some research and development tasks in the robot control area for such collaboration have been outlined in this presentation and the following is a short summary of the R&D directions that have been discussed:

- Sensor-based robot control for safe interaction between human and robot and simultaneously for high performance control of low cost robot mechanics.

- Sensor-based human–robot interfaces for intuitive robot programming and cell calibration.

- Efficient scalable software architectures for interactive robotics.

- Tools to be used in robot installations for automatic identification and tuning of model-based robot control parameters, especially for highly modular robot mechanics.

- De facto standards and easy to use tools for planning, optimization, configuring, calibration, programming and reconfiguring of robot automation systems. Important concepts could be Plug and Play for both virtual and real components and knowledge databases for process- and automation deployment.

- Further integration of process control into the robot controller, especially in applications where force/torque sensors and 3D vision will be used.

2.4 Analysis on the welding task

It represents the task structure and welding tasks [9], to be performed with the movement of the welding torch along the predefined welding trajectories. The set of welding trajectories look like U shapes; hence this is called the U-shaped trajectory welding. The overall process of such U-shaped trajectory welding is divided into the initial positioning and the actual welding. Before initiating the welding process, the start and end points of each welding trajectory can be obtained using laser, or touch sensors with certain sensing algorithms. (i.e. the RRXC can use laser and touch sensors together). This is considered to be the initial positioning, with respect to the inertial coordinate frame. After finishing this initial positioning, the welding tasks are performed in the order of left-vertical weaving,

horizontal multi-pass, and right-vertical weaving welding, respectively. The welding has to be carried out along the contacting boundaries; a vertical path of zig–zag motion on the left side is the contacting boundary between the longitudinal stiffener and the transverse web floor. That is, it is logically divided into several segments to support the job of programming and motion planning of the robots. In addition, there are two reasons for the zig–zag motion in the weaving, which are, (1) it is needed in order to reduce the number of times of multi-pass horizontal welding, if it requires a wide range of gap, and (2) it is also needed to prevent running down of a weld ment in the vertical welding.

It is supposed that many kinds of U-shaped welding trajectories may currently exist with respect to the combination of the positions and dimensions of the bracket, collar plate, and scallop. The required dimensional ranges of each segment of the U-shaped trajectories are typically defined as follows: (1) the height of the longitudinal stiffeners is in the range

250–650 mm, (2) the width between two longitudinal stiffeners is in the range 630–1050 mm, (3) the thickness of the collar plate is up to 35 mm, (4) the radius of the scallop is in the range 50–100 mm, and (5) the length of the bracket is up to 500 mm. Additionally, energy sources of 3-phase electricity supplies of 220 V, and pneumatic power in the range 5 bar to 7 bar can be used in factories.

Design: An intelligent welding robot [9]

To guarantee its straightness in driving along the predefined welding trajectories by holding the guidance wheels against the web floor but the guidance wheel is not acceptable for use in the RRXC, whose degrees of freedom in the end-effector are represented as six-DOF comprising three prismatic and three revolute joints. The main reasons for this are as follows: (1) it may lead to interference between the manipulator and the guidance assemblies in the U-shaped trajectory welding and initial positioning tasks, and (2) it may not guarantee straightness in bi-directional multi-pass welding, and only proper unidirectional welding can be ensured because of the inclined manner of driving against the wall. Naturally, this has been regarded as the most challenging subject in the mechanical design, since it may lead to negative influences in the welding quality if it does not work well.

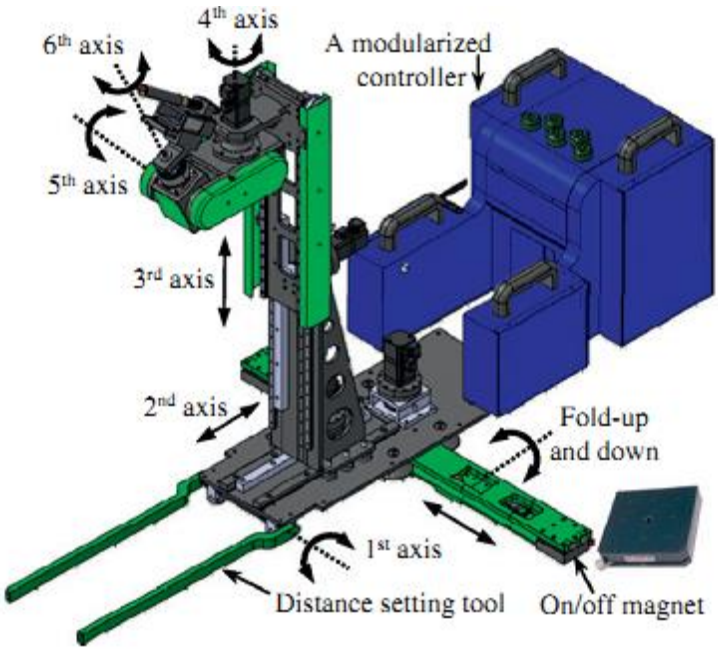


Figure 2.4: View of the developed RRXC, virtual mock-up with representations of dimensions, names, and number of axis[9]

The RRXC is composed of a 6-axis welding manipulator and a 6-axis modularized controller. And, the total weight of the system is 60 kg with the 6-axis manipulator weighing 45 kg and the modularized 6-axis controller weighing 15 kg. (1) 6-axis welding manipulator and positioning devices: the 6-axis welding manipulator is composed of three prismatic axes and three revolute axes, which are driven by AC servo motors. The first axis is driven by a rack and pinion mechanism in a parallel direction to the transverse web floor. In order words, it can make the entire body of the RRXC move on the rack, which is composed of three parts, and is connected by hinges with each other. Moreover, the total length of the racks can be changed by folding up from 760 to 356 mm. The second axis is also driven by another rack and pinion mechanism, in a perpendicular direction to the transverse web floor. In particular, the third axis is driven by combinations of a pulley, a timing belt, and a telescopic mechanism, in the vertical direction. It has three overlapping sliders, namely a multi-slider system, for elevating the welding torch from its rest state, as shown in Figure 2.4. As a result, it has a stroke of 750 mm in the vertical direction, with respect to the bottom plate. The fourth and sixth axes are the yawing and rolling axes which are directly driven by servo motors through harmonic drive

systems. The fifth axis is the pitching axis driven by a pulley and timing belt combination. Figure 2.4 also shows the design of the end-effector, which consists of a laser displacement sensor and welding torch. This is connected to the main sixth axis of rolling through a shock sensor.

6-axis controller [9]: The controller hardware consists of a main controller and a welding machine controller. The main controller, which is mounted on the mobile welding robot, consists of a CPU board, a motion controller, six AC servo motor drivers of absolute encoder type, a flash disk, some relays, power modules, and a power distributor. The welding machine controller, which is mounted on the welding machine located on the outside of the double hull structure, controls the welding machine. The communication between the two controllers is made via the RS485. The motion controller, and servo driver, applied to the RRXC are commercially available from M/S Yaskawa of Japan, and are used because of their reliability in such hazardous environments. Three servo drivers are symmetrically arranged on both sides of the controller, and the rest are arranged in the middle-rear section of the controller to minimize the interference with the end-effector.

Another challenging issue of the main controller is that it needs to be modularized for providing the portable function to enable it to carry out the tasks and to be maintained within the enclosed structures. Thus, in order to make the mechanical separation between a 6-axis robotic platform and a modularized controller possible, all the connectors are embedded on the top plate of the controller, and the two parts are bolted together. Hence the mechanical separation becomes quite simple by disjoining of five-connectors and four-bolts.

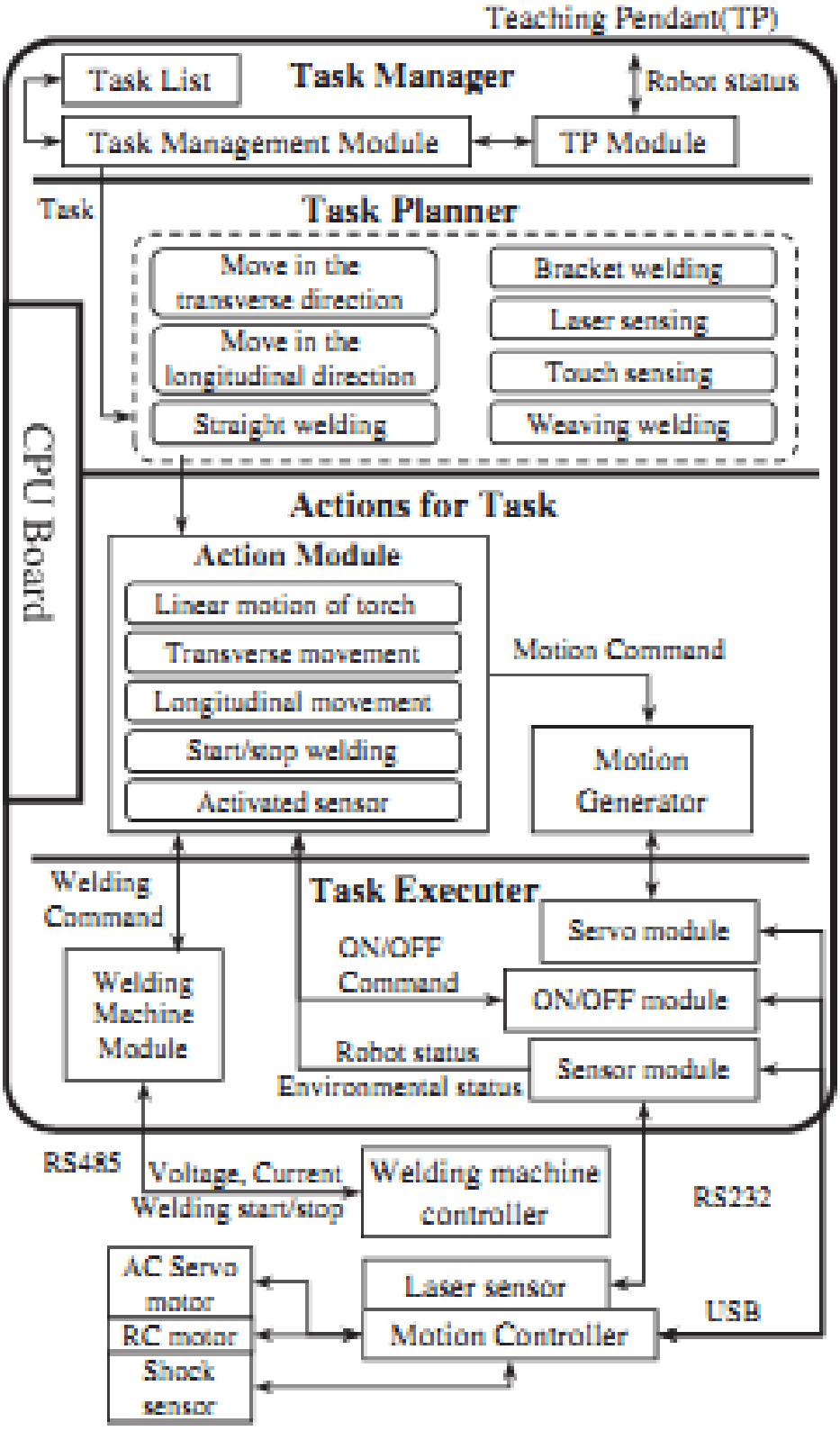


Figure 2.5: The four layered architecture and the modules of the RRXC [9]

Open Robot Control Architecture {ORC} [8]

Firstly the choice of basic principle for architecture is justified, and then the architecture as such is presented. This is followed by discussion of some implementation issues which should also help to clarify the basic concept.

Present Status

From the perspectives of various categories of programmers who need to configure or program industrial robot systems, several programming situations have been identified. When solving a specific application problem, it may need to modify the system in several ways (control laws, operator interfaces, etc.) requiring different types of competence. Assume that only one user type for each type of required competence, each user type viewing the control system in a certain way. Unless the system is carefully designed, any particular such view will be unnecessarily complex (involving a variety of computers, programming environments, special restrictions on use of interfaces, etc.). If it instead base the architecture on properly selected user views, it is more likely that programming can be done conveniently. A major difference from other current approaches is that the external view, rather than the internal implementation, is the primary matter. A view may map well onto internal modules based on some software paradigm or abstraction principle, but this is not an essential criterion.

Software layers

The Open Robot Control (ORC) architecture is based on layers and typical users. The layers are characterized by the following:

- The layers are dedicated to programming cases requiring a certain type of competence. Within industrial production development, such a separation of concerns is crucial for system improvements.

- The servo control has been split up for control engineering reasons. The motion control layer coordinates and commands arm controller(s) and motor control of

external axes. The popular research topic of advanced feedback control of robot motions is encapsulated by the arm control layer.

- The intermediate level has a specific layer for application-specific motion control, admitting more general and advanced control features than other systems do. Handling external sensors, or internal servo signals exposed by the motion control layer, in the application layer permits faster enactment and higher performance than user-level control does.

- The system programming level of other systems is mainly covered by the executive layer in ORC, a layer that also serves as a holder of the robot programming language, which is fixed in other systems.

- There is both an on-line and an off-line programming layer. These are uniquely integrated on an equal level basis. The need for transformation of robot program, opposed to the simple down-load/upload used today, stems from differences in the way work-pieces are preferably referred to in the on-line and off-line programming cases.

- Task-level features are today usually implemented on top of off-line systems. Such features also define a higher-level user interface or programming environment. Therefore, task-level programming has its own software layer in ORC.

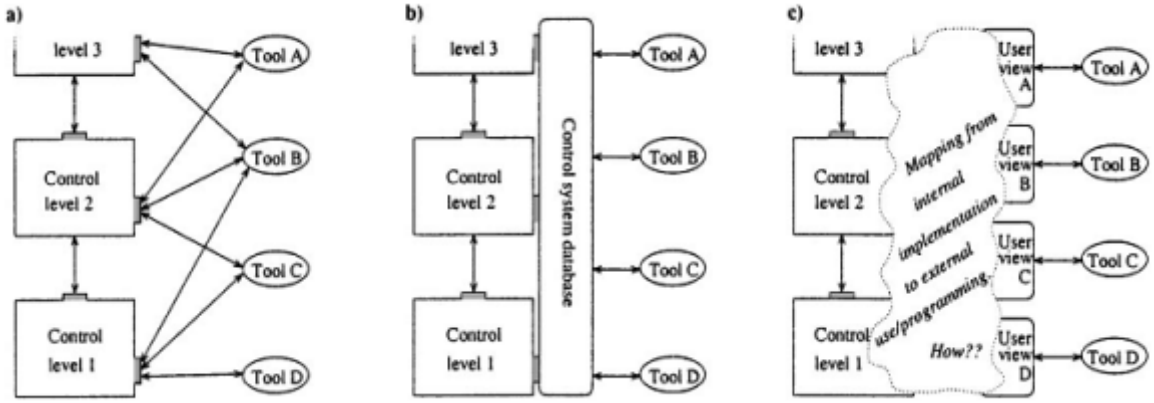


Figure 2.6: Different ways of connecting user interfaces to multi-level control systems: (a) traditional, (b) database, (c) decoupled [8]

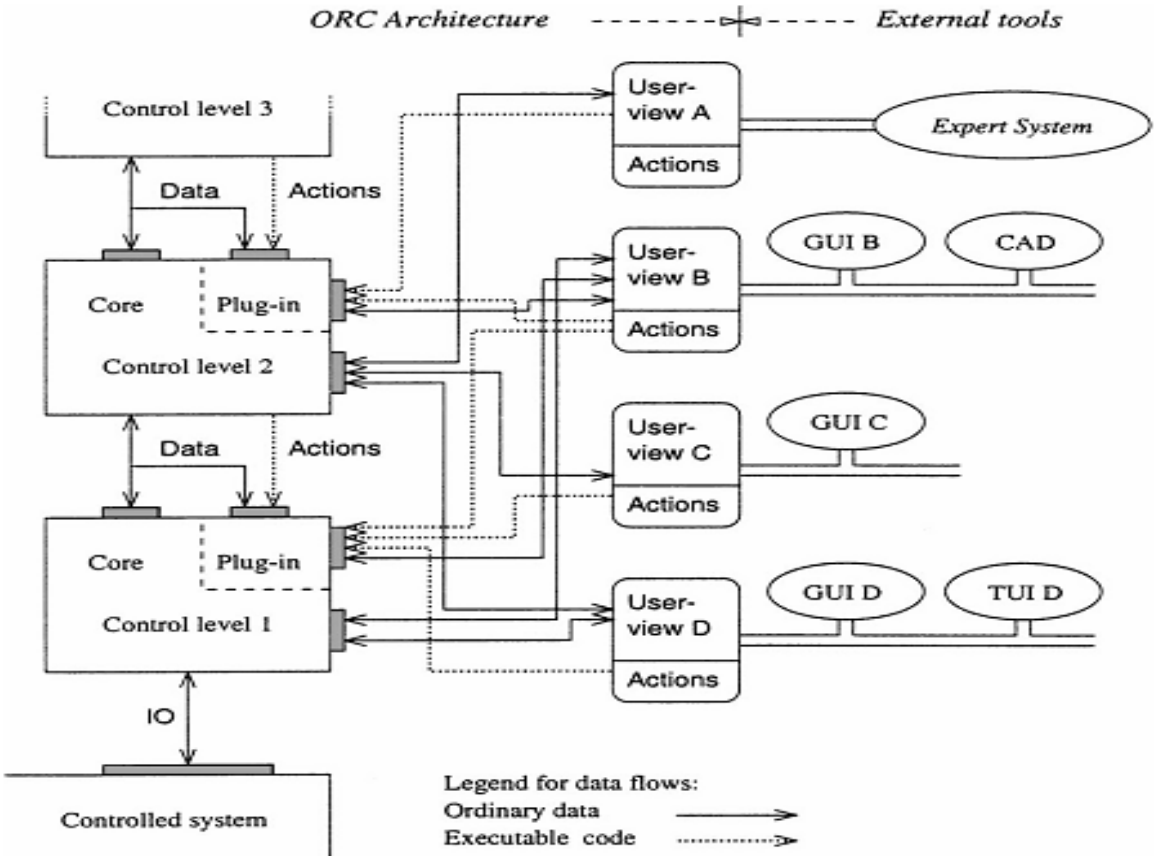


Figure 2.7: Example of separation between implementation layers and user views which exposes the functionality needed for certain users and tools. [8]

ORC layer	Encapsulates	Typical programmer	Typ. exec.	
Task level programming	Automatic programming from design data	Implicit from work–piece design (not possible today)	Interpreted	Host comp.
Off–line programming	Programming without use of robot	Robot programmer with computer experience		
On–line programming	Programming with use of robot	Production engineer or robot operator		Embedded system
Executive	RPL and control system interface	Computer programmer and exp. application engineer	Compiled	
Application control	Application specific motion control	Experienced application engineer		
Motion control	General control of workcell motions	Control engineer		
Arm control	Arm specific motion control	Robot control engineer		
Motor control	Control suitable for impl. in distributed hardware	Servo control engineer		

Figure 2.8: Users and properties of software layers/views in the ORC architecture [8]

End-user programming

The aim that standard industrial robots should be possible to use as components for so-called intelligent robot control implies that task-level programming principles should be superimposed on explicit robot programming tools. Task-level programming facilities can of course be accessible directly from an on-line programming tool, but its software should rely on off-line programming. The reason is that off-line and task level programming have the same need of abstract world modeling, whereas online programming utilizes the physical equipment which results in different requirements on the programming environment. The integration of the on line (tangible) and off-line (abstract) interfaces is a topic of its own, including transformation of robot programs. Superimposing task-level programming on off-

line programming results in the set-up. Note that even if task-level and off-line programming are based on the same tool (IGRIP in our case), the architecture specifies that the task-level features should expose a uniform view to the user.

System-level programming

Properties of computer programming also appear within robot programming. On a system-level this means implementation of (robot and process independent) libraries, and conventional implementation of drivers for IO devices and sensors. This is simply a matter of (computer) programming appearing at any level of the system, and it is therefore not further discussed here. More important, system programming (from an end-user point of view) deals with aspects of specific robot functionality and control of the physical world. We prefer to separate these issues into two categories: (1) implementation of robot-specific libraries, robot programming tools, and robot programming languages; and (2) implementation of application-specific motion control. From an engineering point of view, the latter—i.e. the tailoring of motion control—requires control software, at least for the motor control, are designed as integrated activities to facilitate price-performance optimization. Thus, the software naturally takes on a structure that reflects that of the hardware, whereas programming is performed by the implementers of the system. This implies that the user views of the architecture map well onto an object-oriented design of the motion control system, which in turn reflects the structure of the physical objects.

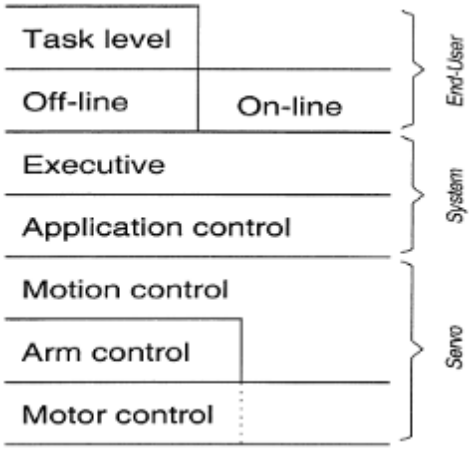


Figure 2.9: The Open Robot Control (ORC) architecture [8]

Servo-level programming

On the level of servo programming, a persistent feature is that software and hardware modules are characterized by their very close relationship. Hardware and software, at least for the motor control, are designed as integrated activities to facilitate price-performance optimization. Thus, the software naturally takes on a structure that reflects that of the hardware, whereas programming is performed by the implementers of the system. This implies that the user views of the architecture map well onto an object-oriented design of the motion control system, which in turn reflects the structure of the physical objects.

Arc-welding control and programming

Of the several needs for intermediate-level programming within arc-welding applications, the following situations have been studied:

Basic functionality: A basic application package for arc welding includes control and programming features for waving motions and control of the welding equipment and interfaces to common types of welding equipment. The Arc Ware package [5] developed within ABB Robotics is an example of well-designed software for this purpose.

Path tracking: Available industrial systems usually also support weld-seam-tracking using a laser-scanner sensor. The sensor is then integrated with the motion control system, but this can so far only be done by the robot manufacturer (because available systems do not provide an open application layer). The path tracking control problem includes estimation of the weld seam based on (often very noisy) data from the laser-scanner, and control of robot motion in such a way that the seam is tracked. The stochastic nature of the problem and the desire to perform high-speed welding of thin-sheet metal without losing track of the seam necessitates a stochastic control approach.

Welding-specific functionality: Arc-welding is a quite complex electrical, mechanical, and chemical process, which can be influenced by voltages, currents, and weld-tool motions. For special materials, or for welding with special demands on quality or productivity, special welding principles have been developed [5]. Sensing and control (of voltages, currents, weld-

joint geometry, etc.), and close interaction with the robot motion control are of key importance.

Task-level programming [8]: Even if on-line programming is claimed to be superior concerning operator adjustments, it should be borne in mind that the initial programming of, for instance, advanced arc-welding programs may often be done better off-line. That is because CAD data for the work-pieces can be used for the definition of welding paths and end-effector orientations. The off-line programming system also provides a platform for task-level programming using knowledge-based techniques. So far this does not imply any need for intermediate-level control, but our desire to use 'hand-crafted agents' does. Special welding techniques according to the previous item should be possible to load and refer to from the task-level system.

A thorough review indicated that the proposed control system design should suit the basic functionality and path tracking needed, but implementation of those parts has been done within our project. As industrially available systems can handle such items, these aspects

are omitted here. Instead, let us focus on the desired 'welding-specific functionality' and on the 'task-level programming'.

Thus, to make a system admitting incorporation of welding control at an intermediate level (here, in the application layer), and an on-line connection to a task-level programming system. The welding control should be possible to define and install from the task-level system, and it should be possible for motion commands from the task-level system to utilize the loaded welding skill.

The purpose of software architecture [5]

Software architectures have received much attention within robotics research [5]. One reason is that when robot controllers (e.g., for space applications) become more and more complex, abstractions and software structures are introduced to cope with the complexity. In other words, the purpose of the architecture is to make the implementation of the system feasible. In industrial robotics the software is complex and various functions must be tightly coupled to achieve efficiency. However, the implementation complexity is not worse than that which can

be handled by proper software engineering methods such as an object-oriented design. However, the variety of user interactions in flexible manufacturing systems indicates that user views of the system should constitute the basis for the architecture. This is a completely different approach that should not be confused with implementation architectures.

When suggesting a new embedded control system, there will always be alternative means of implementation. Take, for instance, some advanced process control systems. First, such a system can of course be used to control a robot, but will the desired performance, cost-effectiveness, programmability, and flexibility be achieved? Secondly, when designing servo control using process control systems, will use of a specific system and its special language etc. be appropriate for interfacing to stand-alone servos? In conclusion, almost anything feasible can be implemented in any system, but specific application demands are typically not taken into consideration. Although the desired application software structure may sometimes be accomplished, performance specifications are not met simultaneously. “Layered systems are not useful without a detailed specification”

Layered systems are perhaps most common within computer communication. Within computer and telecommunication applications, it is crucial that the specification of the layers is complete in all its details. It must be possible to integrate components and layers from different vendors, and the layers reflect the implementation.

Here, the layers reflect user views of the system. Detailed internal interfaces could of course also be developed, but that needs to be done in collaboration with major vendors and/or standardization organizations. Otherwise, the industrial impact would be too small. On the other hand, we claim that earlier and current standardization of robot interfaces on high, intermediate, and low levels are inappropriate. Because this work is not devoted to some new programming language, it does not depend on a detailed standard. It is usage of the principles proposed that yields the benefits. Standards come with maturity! “A robot controller is just another PLC block”

Process control systems and PLCs (programmable logic controllers) often control motions usually via dedicated servo controllers containing the drive electronics and the low-level feedback control. Process controllers are typically programmed by combining and connecting

PLC blocks into a block diagram defining the control program. A servo controller can then be encapsulated in such a block. What then is a robot controller? In simple and less demanding cases, a robot control system is simply a multi-axis programmable servo controller. For the reader whose experience mainly derives from such applications, it may be hard to understand why robot control should be such a big issue; it is just another PLC-block.

However, investigating demanding use of industrial robots, the needs of motion descriptions, operator interactions, non-linear and variable structure control, and computing efficiency clearly show that robot control requires its own control techniques.

A system that is open allows the user or system manager to change or add certain internal components of the system. In practice, systems are a mixture of open and closed parts [7]. The open parts can also be open in many ways. For example, consider a robot control system with a replaceable trajectory generator. Such a system can be claimed to be open. However, the interfaces to the software component (the trajectory generator in this case) could be so rigid that only the algorithm can be replaced. In other words, structural changes involving, for example, new types of interaction with the servo control are often not possible. Therefore, an open system should be reviewed concerning the type of changes possible, and what degree of flexibility that implies. However, it should be borne in mind that there may be safety and proprietary reasons for keeping parts of the system closed.

The Open Robot Control (ORC) [8] architecture defines multiple software layers for different types of users and programming situations. Use of the functional operators turned out to be a key mechanism for implementation of the ORC architecture. It made the user-view concept applicable, for instance, by allowing tight control loops to be conceptually defined within one (high-level) view and then efficiently carried out by the run-time services of another (low-level) view. This proved to work even down to the lowest level of (motor) control, and even when industrial demands in terms of efficiency were considered. Nevertheless, ORC also supports high-level abstractions, and several other architectures can be achieved as special cases.

2.5 Gaps in Literature

After literature review it is observed that in the field of automation, most of the development is made but still there is lot of scope to impart flexibility in existing systems. Not much attention is paid to develop systems with considerable amount of flexibility. Here, flexibility deals with response of the system to produce desired changes in the output with the respective changes in input. Even now, robots are in continuous demand to carry out specific tasks with little expectation about its flexibility to perform variety of tasks. And in the systems where flexibility is provided, there is no such verification tool like simulation is available to proceed the task confidently. Also there is long way to go as per customization in hardware is concerned because without it, whole of the flexibility and verification features are of no use if process is impractical.

Presently, solution to given problem can be obtained with the involvement of huge capital which is not feasible for all kind of industries. Therefore there is need to develop economical solution to the given problem so that it which can be made feasible for different sections of industries. Keeping this factor in mind, the further work is carried out in this thesis.

3. PROBLEM DEFINITION

3.1 Case Study for Implementation

Auto industry is very large consumer of robots. In this era of competition, with the continuous and very frequent change in auto technology, there is the need arising to provide flexibility in robots at shop floor so as they can adapt themselves according to the frequent modifications.

Cylinder walls can become very worn or damaged from use. In such cases the use of a sleeve or cylinder liner can restore proper clearances to an engine.

Multi cylinder heavy engines are used in railways, to recondition these engine blocks, there is need to fill the material along the circular shape of the block which is a cumbersome task to be perform manually. Because if the given task is to be carried out by human in case of 12 cylinder engine, it results in over fatigue of labour and poor quality weld which becomes a challenge for quality control. Therefore, the solution to the current problem is the use of robotic arm. To carry out this task, robots with prismatic and cylindrical joints are the most obvious choice but in case of industry already having articulated arm robot, the given task is conventionally not recommended. So, instead of investing huge amount on new compatible configured robot, the available robot must be adaptable to perform the desired task efficiently.

During welding along the inner periphery of cylinder, path to be followed by the tool tip should be expanding helix because with the continuous consumption of electrode, it also needs to be advance along the thickness of the cylinder to maintain arc between the cylinder and electrode.

Before implementing the developed algorithm of trajectory on robot hardware, simulation of the same must be carried out in virtual environment. With the help of this virtual environment, path followed by each and every joint while following helical trajectory can also be known well in advance and if in between during motion any constraint is violated, it can be rectified and corrected before implementing it on hardware which helps to avoid any damage and loss. After passing through various stages of programming verification and CAD simulation, The values obtained from inverse kinematics are used to drive the motors of rhino robot to trace the required trajectory.

3.2 Practical limitations of Rhino Robot for Selected Case Study

Structure Limitation: The Rhino robot is an educational robot with jointed arm configuration which is constrained to move within its range of allowable joint angles. Starting from the base, every joint is restricted to move according to joint motion constraints which define the joint space of robot. Due to which domain of the path to be followed by the robot is also restricted which defines its reach and work envelope. According to the dimensions of structure links, the reach, stroke and joint limits [18] are:

Horizontal reach and stroke: 62.23 cm

Vertical reach and stroke: 88.27 cm

Motor "F" Body Rotation - 360 degrees

Motor "E" Shoulder Rotation - 250 degrees

Motor "D" Elbow Rotation - 270 degrees

Motor "C" Wrist Rotation - 270 degrees

Motor "B" Gripper Rotation - +/- 7 revolutions

Therefore all these structure limits must be taken into account while developing any algorithm of path planning.

Controller Limitations

The rhino robot controller is a basic 8 axis controller which can interact only with 8 TTL level input lines and can control only 8 TTL level output lines. It can control only two auxiliary ports that provide 20 volts each at 1.5 amps. All input lines are pulled up to TTL high. Output lines provide TTL level signals at 100 milliamps. The auxiliary ports can control only two more motors or relays as needed by the experimenter. Controller is not adaptable to change or edit the control language and is dedicated to perform specific jobs only.

RoboTalk (Control Language) Limitations

A robot control platform is designed to provide a control interface for mobile robots and sensors, it currently has the following limitations:

1) Player server is designed to directly access the robot/sensor drivers, which is not a good option for black-box components or changing hardware.

2) Player lacks the ability to define and access individual joints, links and control frames on a humanoid or a kinematics simulator. It can be argued that this capability can be added to Player, but this is far from a trivial addition.

3) The Player server cannot issue multiple commands to different motors in a robot at the same time. In androids, it is important to have the ability to issue simultaneous commands because it allows the possibility of whole body coordination.

4) There is no mechanism in Player to customize Panic responses to prevent disastrous outcomes on different types of expensive robots. This modification again is not trivial, because the communication model must understand (as opposed to just communicate) the concept of Panic in order to pre-empt other actions.

5) The communication model between server and client in Player was designed in blocking mode, meaning the client cannot process a new command before it receives an acknowledgment about the previous command. Finally, there are no command buffers or return caches built within Player. Thus, a motion sequence sent over long distances will not replay in the same way when it is sent locally.

This thesis deals with all the limitations discussed above so as to perform the desired task considering all these constraints.

4. IMPLEMENTATION

4.1 Robot Kinematics and dynamics

The study of motion can be divided into kinematics and dynamics. Direct kinematics refers to the calculation of end effector position, orientation, velocity, and acceleration when the corresponding joint values are known. Inverse kinematics refers to the opposite case in which required joint values are calculated for given end effector values, as done in path planning. Some special aspects of kinematics include handling of redundancy (different possibilities of performing the same movement), collision avoidance, and singularity avoidance. Once all relevant positions, velocities, and accelerations have been calculated using kinematics, methods from the field of dynamics are used to study the effect of forces upon these movements. Direct dynamics refers to the calculation of accelerations in the robot once the applied forces are known. Direct dynamics is used in computer simulations of the robot. Inverse dynamics refers to the calculation of the actuator forces necessary to create prescribed end effector acceleration. This information can be used to improve the control algorithms of a robot.

In each area mentioned above, researchers strive to develop new concepts and strategies, improve existing ones, and improve the interaction between these areas. To do this, criteria for "optimal" performance and ways to optimize design, structure, and control of robots must be developed and implemented.

Forward kinematics is computation of the position and orientation of robot's end effector as a function of its joint angles. It is widely used in robotics, computer games, and animation. The reverse process is known as inverse kinematics.

Inverse kinematics is a subdomain of kinematics, which is of particular interest in robotics and (interactive) computer animation. In contrast to forward kinematics, which calculates the position of a body after a series of motions, inverse kinematics calculates the motions necessary to achieve a desired position.

Examples of problems that can be solved through inverse kinematics are: How does a robot's arm need to be moved to be able to pick up a specific object? What are the motions required to make it look like an animated character is picking up an object?

Solving these problems is usually more involved than simply moving an object from one location to another. Typically, it requires the translation and rotation of a series of interconnected objects while observing limitations to the range of motions that are physically possible. A robot might damage itself if mechanical limitations are disregarded; an animation looks unrealistic if a character's hand moves through their own body to pick up an object located behind their back.

4.2 Inverse Kinematics of Rhino robot

Here we are dealing with five axis Rhino Robot, to control its position & orientation, inverse kinematics needs to be done. In order to develop a solution to the inverse kinematics problem, the desired tool configuration must be specified as input data. The tool configuration is represented by the pair {p, R}, where p represents the tool position relative to the base and R represents the tool orientation relative to the base. Then the tool configuration vector is a vector *w* [10] defined as:

$$w \approx \left[\frac{w^1}{w^2}\right] \approx \left\{\frac{p}{\left[\exp\left({}^{q_n}/_{\pi}\right)\right]r^3}\right\}$$

For which, tool configuration vector approach is used from which all joint angles according to desired positions are calculated.

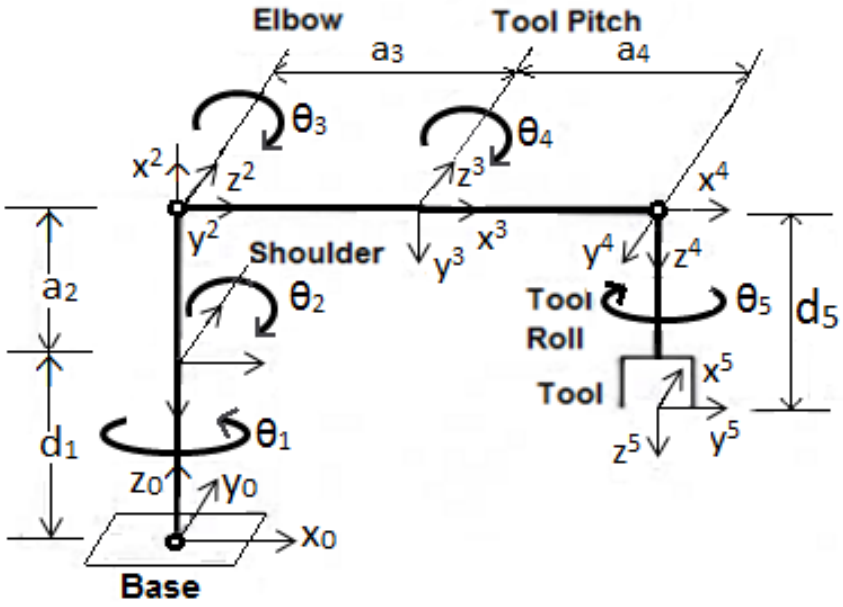


Figure 4.1: Link diagram of Rhino robot [10]

Firstly, line diagram of the Rhino robot is drawn and starting from base, coordinate frame is assigned at every joint up to the tool tip. Then DH Parameter table at home position is made as shown in the Table 4-1 in which the value of α and θ are in degree while the value of *d* and *a* are in cm.

Link	d	a	A	θ
1	24.765	0	0	θ1
2	0	12.065	-90	θ2
3	0	22.860	90	θ3
4	0	22.860	0	θ4
5	12.383	0	-90	θ5

Table 4-1: D-H parameters of Rhino robot [11]

The algorithm for five axis articulated arm (Rhino XR-3) is shown in Figure 4.2.

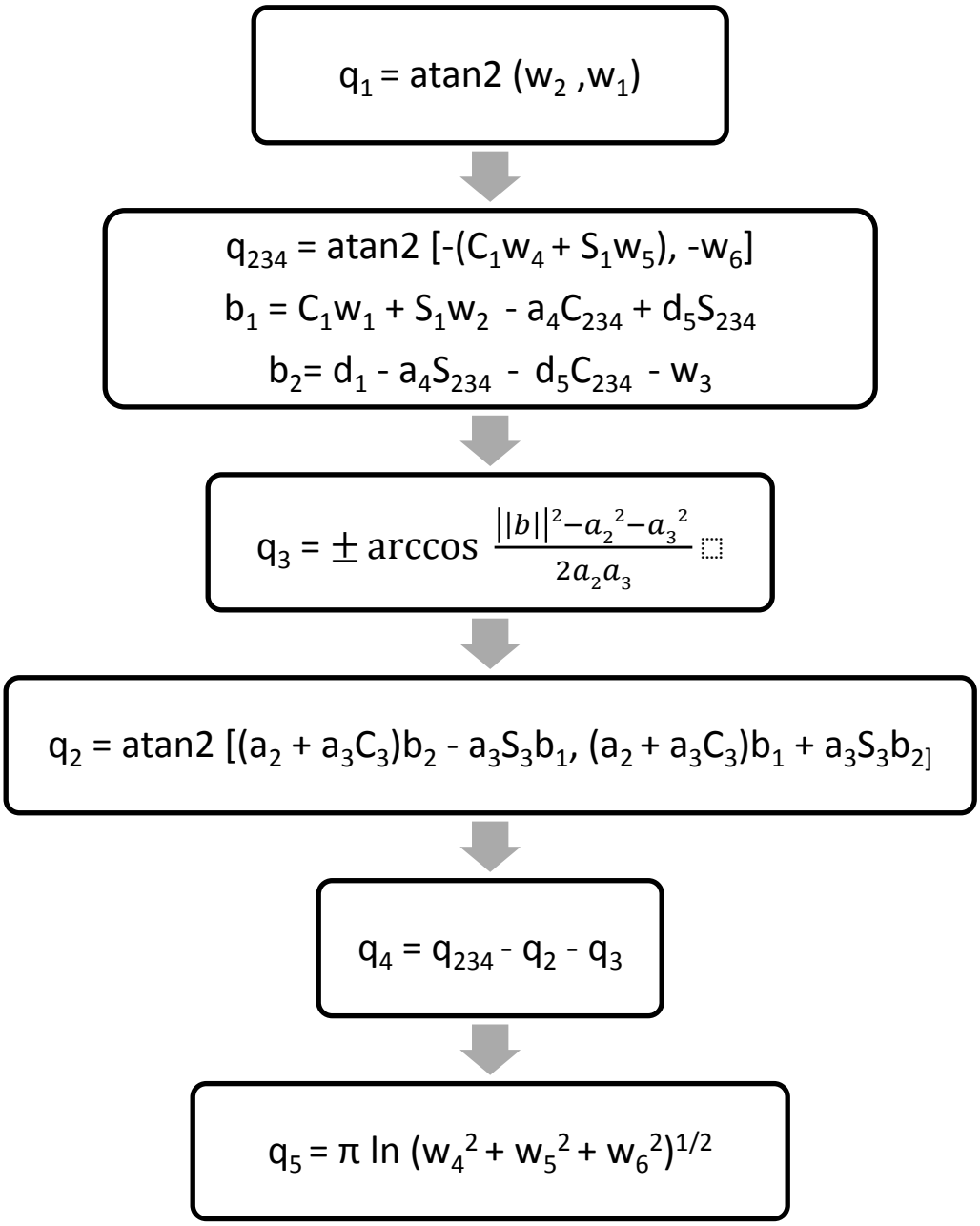


Figure 4.2: Flow chart to calculate joint angles [10]

The given algorithm of the inverse kinematics is applied by keeping in mind the joint constraints i.e. limits up to which motion between each joint is allowable. Every joint has different joint limits according to which work envelope is finalised. The limits for each joint motor are:

Motor "F" Body Rotation - 350 degrees

Motor "E" Shoulder Rotation - 210 degrees

Motor "D" Elbow Rotation - 265 degrees

Motor "C" Wrist Rotation - 310 degrees

Motor "B" Gripper Rotation - +/- 7 revolutions

Motor “A” is used only for opening and closing of gripper.

To get clear idea about work envelope, the reach of the robot must be calculated along with the joint constraints. After measuring, the details about reach are:

Horizontal reach and stroke: 62.23 cm

Vertical reach and stroke: 88.27 cm.

While developing algorithm the joint space constraints of the Rhino XR-3 are taken into account as shown in the matrix form below.

$$\begin{bmatrix} -\pi \\ -3\pi/4 \\ \pi/4 \\ -5\pi/4 \\ -\pi \\ -3\pi/4 \\ -5\pi/4 \end{bmatrix} \leq \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix} \leq \begin{bmatrix} \pi \\ \pi/4 \\ 3\pi/4 \\ \pi/4 \\ \pi \\ \pi/4 \\ \pi/4 \end{bmatrix}$$

Figure 4.3: Joint limits written in matrix form [10]

where q_1, q_2, q_3,q_4,q_5 are the joint angles.

After considering all the joint limits and other limitations, in this case rhino robot is made to follow helical path as per requirement.

The helical path is developed using equation of helical path as follows:

$x = r * \cos(t)$

$y = r * \sin(t)$

$z = n*t$

where x, y, z are the point coordinates along the trajectory w.r.t origin, r is radius of circle, t is increment step and n is the no. of turns

As the task of the robot is to weld along the cylinder throughout its length, the path to be followed must be developed by keeping in mind various parameters:

- Length of electrode constantly decreasing in case of Arc Welding.
- The robot is 5 axis Articulated arm, only revolute joints.
- Link of the lengths are fixed, horizontal & vertical reach cannot be altered with any attachment except by varying end effector length.
- While developing path, consumption of the electrode must be taken into account.
- Angle of Arc Welding lies between 5 to 30^0 from the normal to the surface. Therefore orientation of the tool must vary between this limit.
- During operation while following path, no collision or interference occurs between robot links or work piece.

Considering all this factors, the path generated to weld cylinder with help of given robot becomes helix with increasing diameter.

The helical curve generated becomes path for tool tip. On this curve, transparent cylinder body is also created to visualize movement of the tool during welding. For this, end effector to be used should be of sufficient length so as after holding electrode it can go deep inside the cylinder. Longer end effector increases the reach of the robot but there is limit up to which this length can be extended as extra length rises weight & exceeds joint limits also. Therefore

length of end effector should be selected effectively, this factor also decides the range in which cylinder of different diameters can be welded using this robot.

4.3 Steps of Implementation

Work piece Size and Orientation

The radius, length, position and orientation of the cylinder to be welded is an important consideration to perform the given task effectively. As per the dimensions are concerned, the maximum length of the cylinder should be less than combined length of shoulder and elbow whereas maximum radius should be less than the combined length of wrist and electrode and minimum radius should be sufficient which allows to maintain minimum required weld angle of the electrode.

Position of the work piece should lies within work envelope of the robot and orientation should be such that no interference between the work piece, electrode and robotic arm occurs.

Constraints of axis intersection check

In order to utilise full arm length of robot i.e. to carry out weld up to the boundary of work envelope, the work piece should be oriented in such a way so that while welding the minimum weld angle should be maintained and robot tip able to reach the full length of cylinder which is possible only if the centre axis of the cylinder must be passing through the line of intersection between the waist and shoulder.

Weld rod angle change

As discussed before, to perform continuous welding inside cylinder, expanding helix trajectory should be followed which needs change in angle of electrode at regular intervals. As the robot’s wrist carrying electrode is not having rotary motion like human wrist, therefore synchronization of waist, elbow and wrist is required to trace the desired trajectory.

Weld path direction

The direction of weld path should be towards robot itself because it avoids the need to develop another path which is required to carry out the robot safely out of the cylinder without any interference after welding.

Compensation for consumption of weld rod

With the continuous consumption of electrode, its length keeps on decreasing and simultaneously robot is moving out of the cylinder as in this case, the weld is always started from the outer most point. Therefore to maintain weld arc, with the decrease in electrode length, weld depth should be continuously increasing with the same rate of consumption of electrode.

Steps to maintain weld accuracy

(i) The position of axis of cylinder must be in line with the shoulder axis so as to utilise the full length of robotic arm.

(ii) To maintain the weld arc i.e. to maintain the contact between electrode and cylinder, expanding helical path needs to be followed where the expansion rate depends on the rate of consumption of electrode.

(iii) The work piece should be firmly held and placed within reach of the robot with an orientation which does not force the robot to work below weld angle limits.

Speed Control

To execute the task smoothly, the speed should lies within control limits which ensures continuous weld and absence of additional noise and vibrations which may cause damage to electrode, work piece or robot itself.

Approach and Withdrawal

To initiate the welding inside cylinder, firstly robot tool tip reaches the outermost point which helps to check interference and also verifies the possible motion of the robot during process

before it starts. Then with the continuous weld operation, robot is moving out of the cylinder. It also eliminates the need to develop additional path for the robot to move out of the cylinder.

User defined inputs to program

The helical path is generated with the help of equation in which radius, length of cylinder, no of turns to be made while welding, increment step of helix, all can be given as user input according to the requirement which enhances flexibility of the system.

4.4 Verifying Inverse Kinematics by plotting path from forward kinematics.

Firstly, the path which needs to be traced by the robot tip is defined using mathematical path equations then after solving, the points obtained through these equation are plotted which is defined as theoretically obtained path.

To trace this path by robot, inverse kinematics is carried out which in turn provides sets of motor joint angles needed to move along required path. Then to verify the inverse kinematics algorithm, the given joint angles are used to carry out the forward kinematics which in turn provide the various points to get the continuous path.

The results or path obtained after forward kinematics is compared graphically (as shown in Figure 4.4) with the theoretically obtained path in order to verify and check the precision of inverse kinematics algorithm.

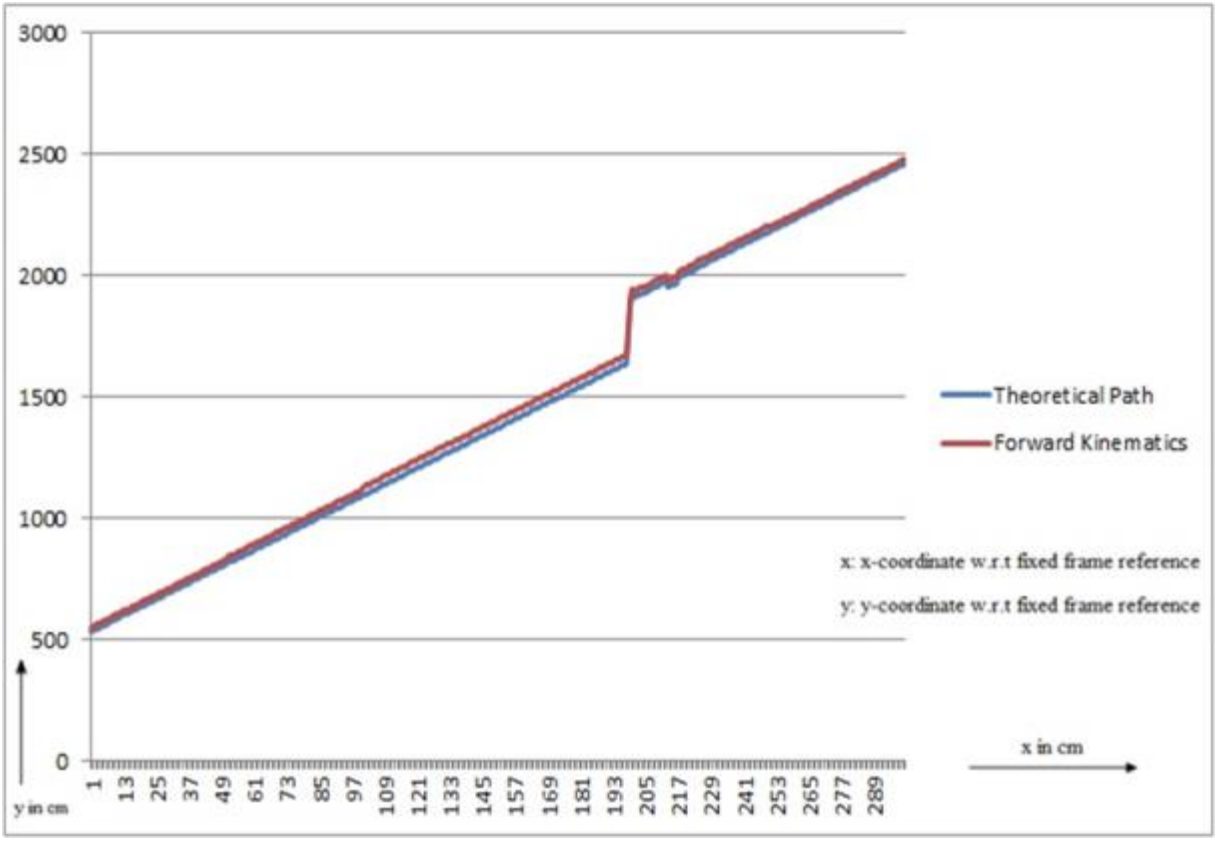


Figure 4.4: Graphical comparison between theoretical path and forward kinematics.

In the above graph, the range of variation is 1.8mm to 3.5mm along y axis. Here, the variation shows that for some portion of the path, algorithm is not as much accurate; this portion mainly lies between two consecutive spirals of helical path where motion is not in a single plane only.

From the above graph it is clear that inverse kinematics algorithm is precise enough to carry out the required task as the variation is very less, the points generated and plotted in this graph are obtained theoretically only after calculations using forward and inverse algorithm with the equations of the path.

4.5 Verification with Custom Built Robot Simulator

Design of Simulator of Rhino Robot Motion using 3D mechanism design in Creo CAD CAE Software

Robotic Simulator built in standard mechanism design CAD/CAE package of mechanism design extension in Creo (ProEngineer) software. With the help of results obtained from the CAD simulator, physical feasibility can be checked and viewed. It helps to predetermine occurrence of any interference between links and body, locking of joints, violation of joint constraints or exceeding approachability of robot. Some limitations which are not imagined or calculated before the execution of path can be find out. With the help of this virtual environment, path followed by each and every joint while following helical trajectory can also be known well in advance and if in between during motion any constraint is violated, it can be rectified and corrected before implementing it on hardware which helps to avoid any damage and loss.

Before implementing the developed algorithm of trajectory on robot hardware, simulation of the same must be carried out in virtual environment. For this, Creo (ProEngineer) CAD software was used to model the whole robot using part & assembly workbenches.

Considering all available limited resources, to perform this task more effectively, the following factors may be considered:

(i) The position of axis of cylinder must be in line with the shoulder axis so as to utilise the full length of robotic arm.

(ii) To maintain the weld arc i.e. to maintain the contact between electrode and cylinder, expanding helical path needs to be followed where the expansion rate depends on the rate of consumption of electrode.

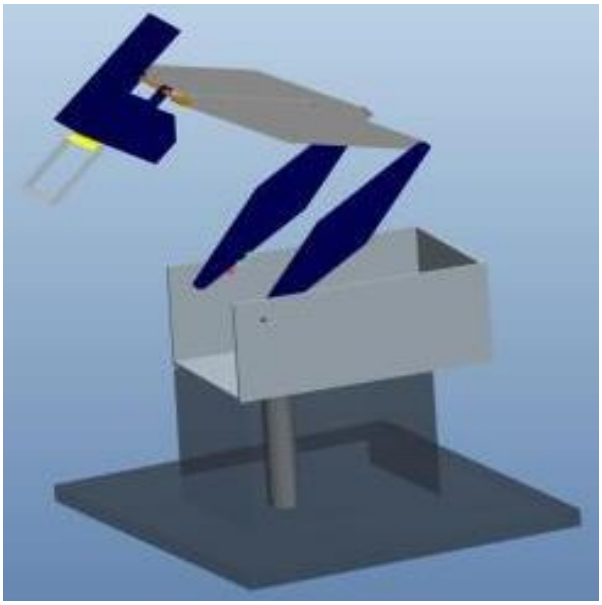


Figure 4.5: CAD Model of Rhino Robot

The complete model is generated as per the dimensions of physical robot and in Figure 4.5, robot is in the HOME position, In this position all the limit switches employed at every joint is pressed. For modeling, each link, base, gripper & rigid bodies are created in Part design as per dimensions then assembled together in Assembly workbench of Pro-E. During assembly, as per physical model, natures of joints are specified. All joints in this model are revolute as mentioned earlier also. Then in Mechanism window by selecting joint axis according to model limitations, the range of the joint motion is given at every joint to create viable simulation. The motion axis window in Mechanism section of Pro-E where the joint limits are specified as shown in the Figure 4.6

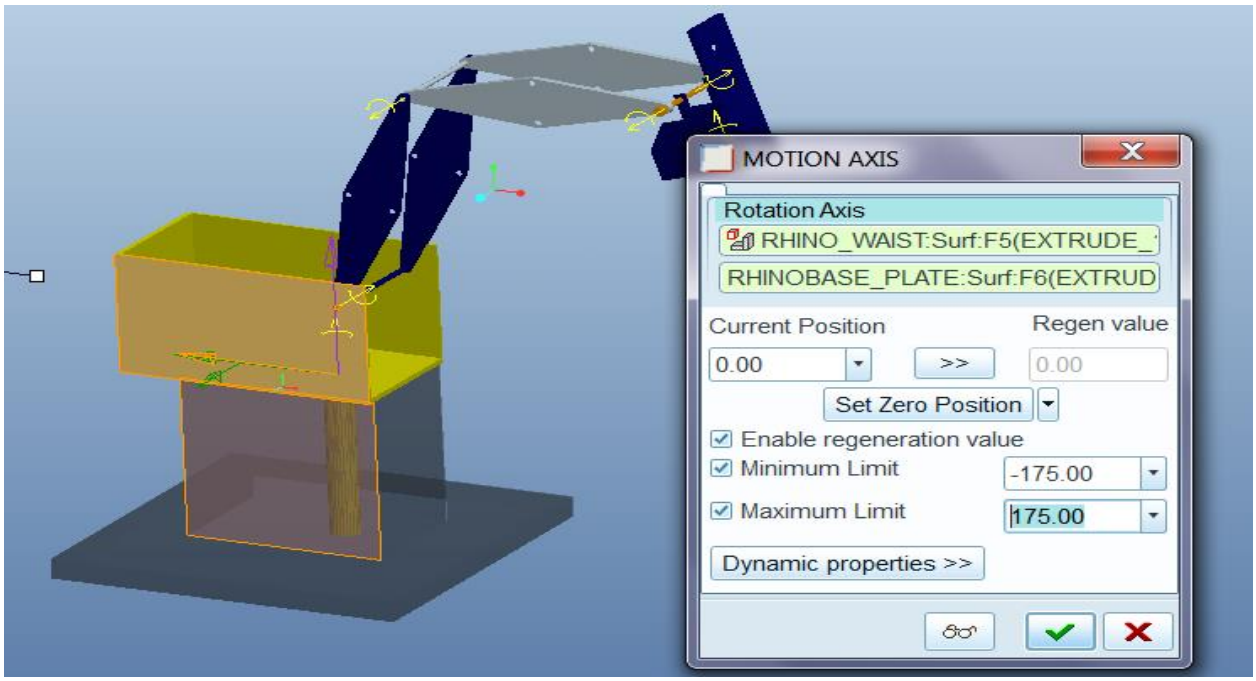


Figure 4.6: Setting Joint Limits

The similar procedure is repeated to set joint motion limit of each joint.

The robot with electrode & selected length of end effector to weld a given cylinder is shown in the Figure 4.7. In this figure, a line is drawn between tool tip & helical curve, This line showing the path which the robot tip follow to reach the point from where the robot will start welding & following helical path which was developed earlier. Through the transparency of cylinder, helix is visible along the cylinder which is the robot trajectory.

The position of axis of the cylinder with respect to the robot can be altered as per convenience of the operation to be carried out. Here only an example is taken to present the idea by taking the position of the cylinder to be welded in an overhead location. Although the cylinder can be placed anywhere in the work volume of the robot with the axis of the cylinder intersecting the vertical axis of the waist motion of the robot.

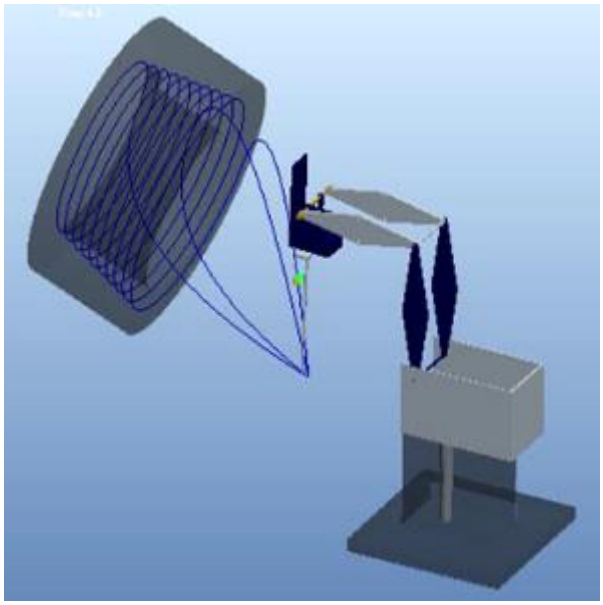


Figure 4.7: Robot with Welding Tool

While following this trajectory, different positions of the robot occurs to ensure motion within joint limits & without interference between rigid bodies. The snapshots of various positions at different instants are shown with the help of Figures. 4.8- 4.12.

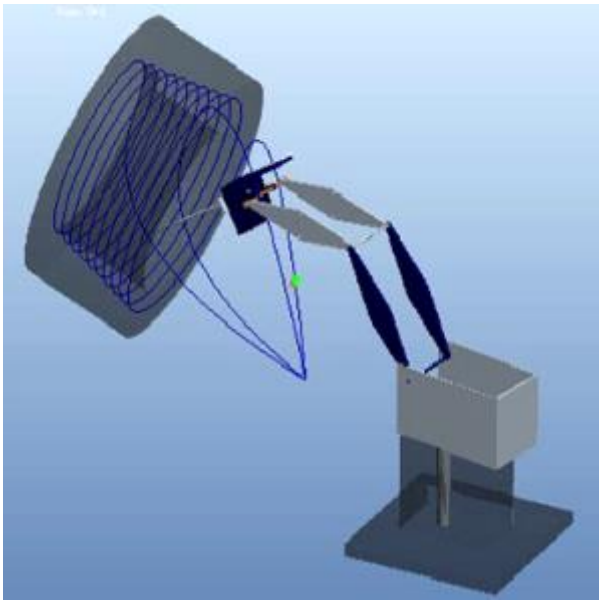


Figure 4.8: Robot welding inside cylinder

In these snap shots, various positions of robot while moving in a helical path is captured. It gives an idea about robot movements while following helical trajectory.

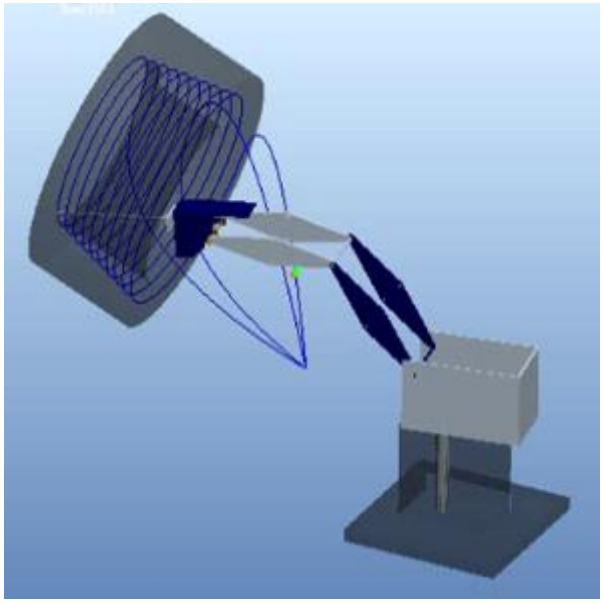


Figure 4.9: Welding along circumference of cylinder.

The Figure 4.10 shows the position while welding on the upper periphery of the cylinder and maintaining weld angle also.

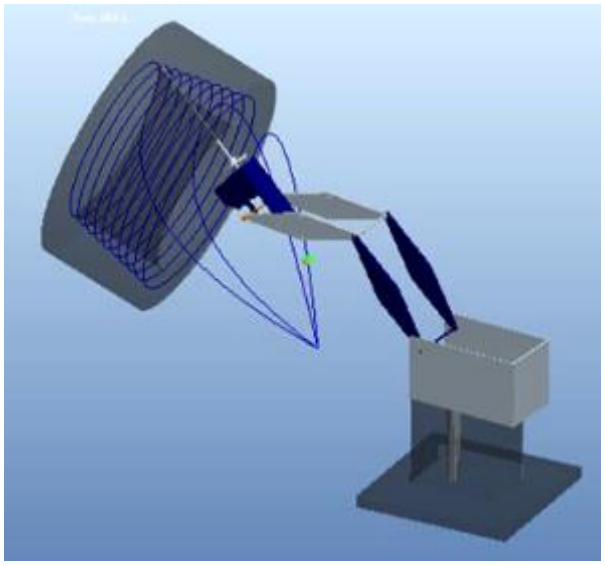


Figure 4.10: Welding along upper section of cylinder

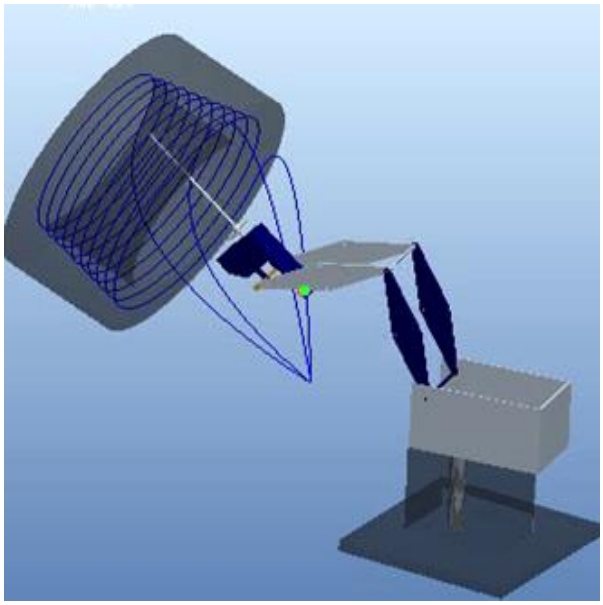


Figure 4.11: Robot adjusting itself to maintain weld arc

During welding, if the size of the weld is large then robot must be able to move out of the cylinder also to get new electrode and resume welding. In the Figure **4.12**, Robot is moving out of the cylinder following predefined trajectory to collect electrode.

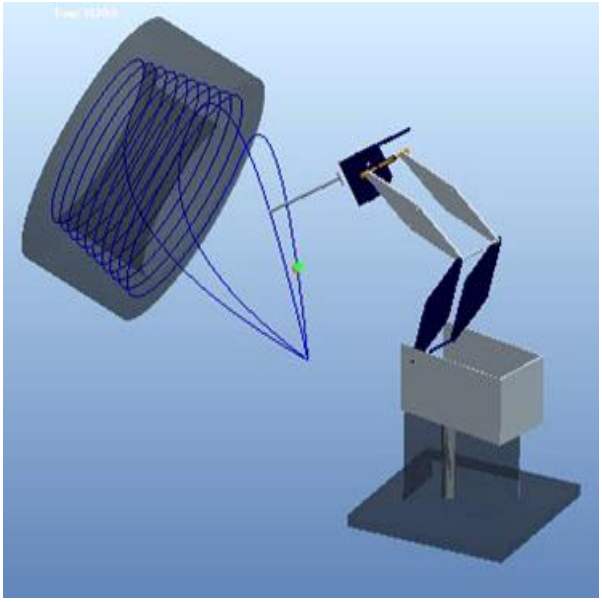


Figure 4.12: Robot moving out of the cylinder

After changing electrode, robot will return to the same point where electrode ends to continue welding. The changing of electrode is not simulated here as the present work is focussed on the path to be followed. It can be easily incorporated if required by providing a withdrawal to a point for electrode change, and a resumption of the path of welding compensating for a new electrode length.

4.6 Verifying Inverse Kinematics by plotting path using custom robot simulator

With the help of simulator, the accuracy of inverse kinematics algorithm can also be cross verified. The various points followed in simulation to trace the given path can be known through export data option in Mechanism window of Creo software.

Then these points are plotted against the points obtained from mathematical equation of the path. i.e. path followed in simulation is compared with the theoretical obtained path.

The graphical comparison is shown in the figure below.

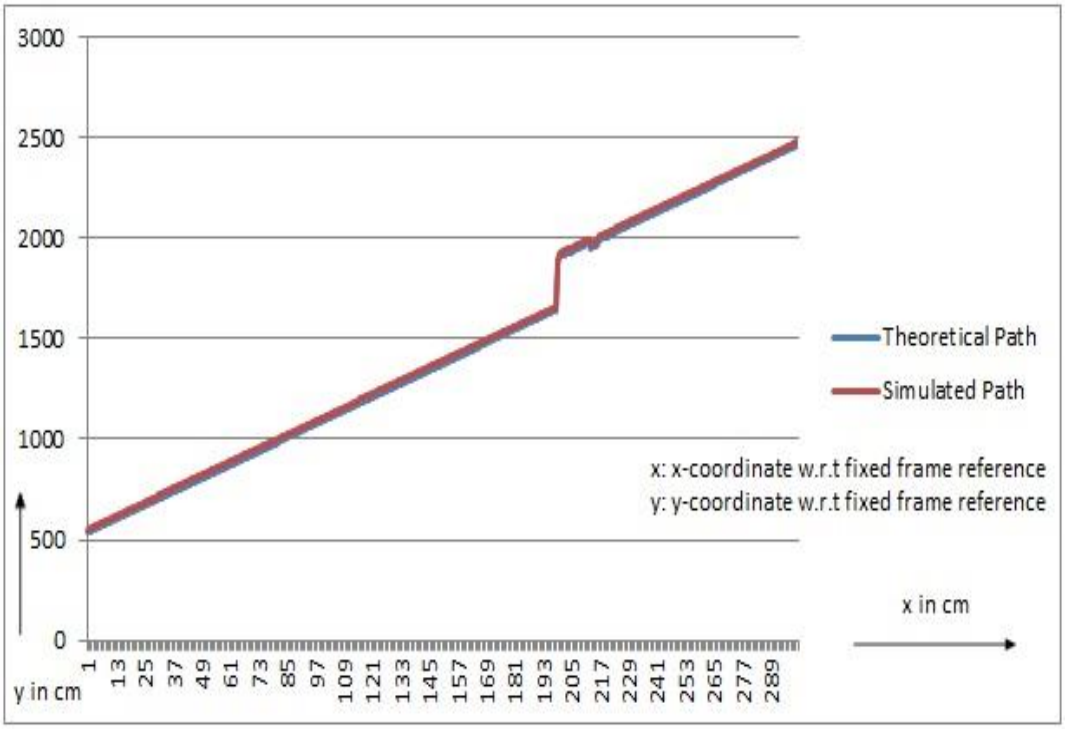


Figure 4.13: Comparison between theoretical and simulated path

From the graph it is clear that variation in simulated and theoretical obtained path is very less. With this result, not only inverse kinematics is verified but also simulator accuracy is also verified. In this case also, all the points obtained are theoretical so their outcome result is also theoretical in nature. Here, the variation is almost uniform throughout and rounded off equal to 1.8mm. The reason for this variation may depend upon dimensional accuracy of the components made in CAD software because the various parts are drawn using integer

dimensions only then these values as link parameters are used to carry out various kinematics calculations whereas the theoretical path is made independent of any kinematics calculations. Still, the obtained results are encouraging to carry out next step of physical experimentation.

4.7 Physical Experimentation on Robotic Arm

After verification by forward kinematics and CAD motion simulation, the values obtained from inverse kinematics are used to drive the motors of available rhino robot to trace the required trajectory. To execute the given task, algorithm of inverse kinematics of rhino robot is written into form of program which is then fed into the PIC controller, retrofitted on the robot, using PIC Flash device. After carrying out calculations using the on board custom program, the controller sends the drive signals to various motors employed at different joints. During execution, time lag occurs but when controller starts carrying out calculations and sending pulses to motors, it becomes a continuous process and robot tool tip starts following desired trajectory.

For more details about the PIC controller and PIC Flash device refer to Appendix.

Limitations of experimental setup

During implementation on the physical robot, large variation from the required path is noted. Although there is large variation in the path followed, due to mechanical errors discussed below, the robot was generally oscillating about the desired path.

The factors responsible for these deviations are:

- Precision and accuracy of physical robot
- Accuracy of calculated joint angles
- Ability of controller
- Mechanical vibrations of structure
- Implemented on educational purpose robot

One position of the robot is shown with the help of snapshot, and comparison with similar postures achieved in simulation is done as shown in the Figures 4.14 & 4.15. It shows the position while welding on the upper periphery of the cylinder and maintaining weld angle also. These show that the algorithm is successful while a better robot would be able to execute it accurately.

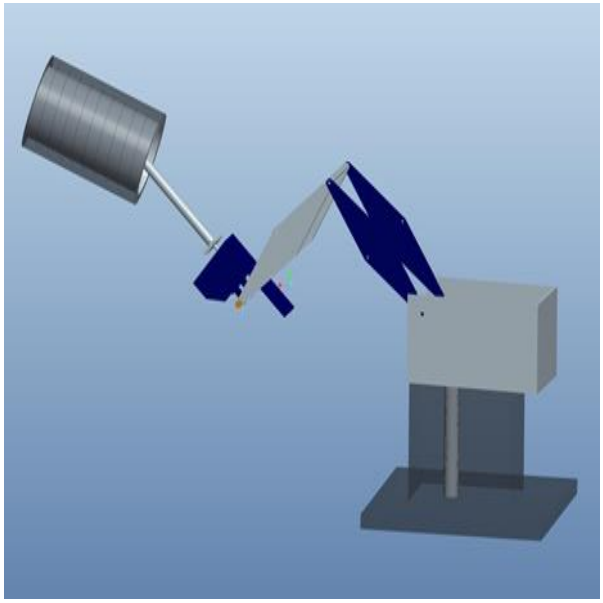


Figure 4.14: Robot welding inside cylinder

Figure 4.15: Robot similar position obtained on hardware

The available Rhino robot is a 25 year old flood effected one. It has backlash error in the chain drives, old vibrating structure, servo motors with vibrations etc. Repairs of the structure and replacement of the damaged controller with a 'PIC' controller was done within economic constraints but the results were not very successful.

The variation are represented with the help of graph shown in

Figure **4.**16

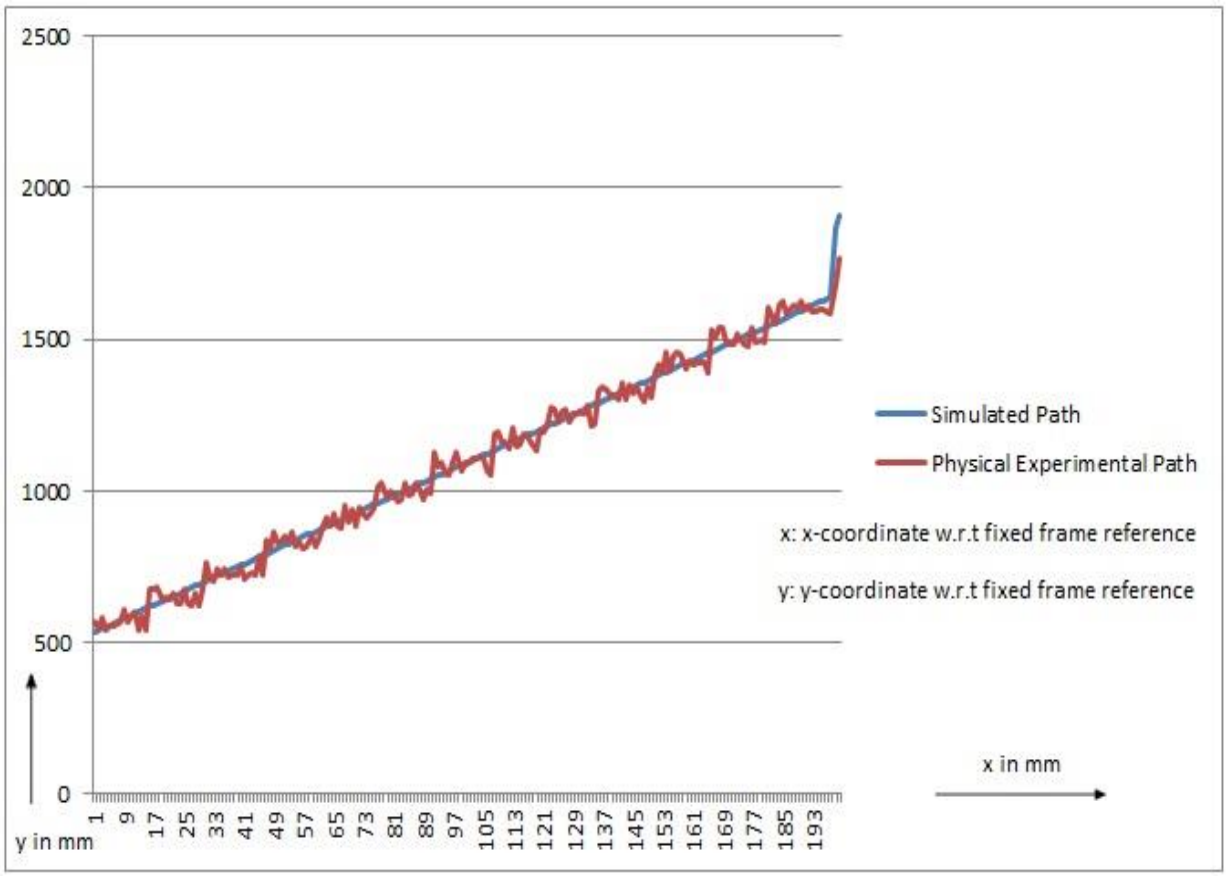


Figure 4.16: Comparison between simulated and actual path traced by robot

From the graph it is clear that path obtained by physical robot is fluctuating around the exact desired path because of few reasons already mentioned. In this graph, the range of the variation is 1mm-42mm. Here, a portion of the path is considered for the purpose of demonstration. This variation can be minimized with a new robot.

5. CONCLUSION & DISCUSSION

From this work, it is clear that within work envelope of robot, different trajectories can be planned, which the robot configuration or supplied programming is designed for. By taking the joint limits & work envelope constraints into consideration, the path can be developed, which can be verified using the custom built CAD/CAE Simulator. The simulator is not restricted for a specific program, it is general purpose for any developed program of the robot.

5.1 Program Conclusion

The program developed in this thesis is valid to follow helical path only and cannot be used directly for general purpose but by using same concept i.e. by combining equation of the required path and inverse kinematics, it can be extended to be used for different trajectories. The program is custom built so can incorporate process parameters like weld rate, tool angle, rate of electrode consumption as per requirements.

From the comparison of path followed during simulation and real time motion it is clear that obtained values of joint angles after carrying out inverse kinematics are not ideal i.e. the curves generated by helical path equation and actual path traced by tool tip is not completely overlapping. The variation or error is caused due to several reasons. One of the reasons is rounding off of results up to two digits. As the program of inverse kinematics was written onto the controller and it compiled and calculated the results for each position the computation speed and capability of the PIC controller was a limitation, as discussed further. If higher decimal places are taken, it takes lot of time to execute, thus increasing time lag. Also to get close results, program of inverse kinematics needs all float variables which consumes double memory as compared to integer type due to which memory limitations also occur. Therefore it restricts the domain of functions and features to be added in the program.

In the present work there was a restriction to the resources and funds available for using a computer interface to the PIC controller which can be incorporated to overcome the limitation in memory and processing capabilities. Once a computer is employed, it will carry out the actions of all calculations and finally transmit required pulses to the controller which will

further transfer them to motors to perform desired action. In this work the concept of custom programming to adapt any robot to unusual and complex paths is developed within the scope of available funds and resources

Some limitations in PIC controller:

The PIC architectures [19] have these limitations:

- One accumulator
- Register-bank switching is required to access the entire RAM of many devices
- Operations and registers are not orthogonal; some instructions can address RAM and/or immediate constants, while others can only use the accumulator.

Therefore all these factors like deviations must be considered while performing it for practical purposes or further more accurate inverse kinematic equations can be developed.

5.2 Robot Simulation Conclusion

Robotic Simulator built in standard mechanism design CAD/CAE package of mechanism design extension in Creo software. With the help of results obtained from the CAD simulator, physical feasibility can be checked and viewed. It helps to predetermine occurrence of any interference between links and body, locking of joints, violation of joint constraints or exceeding approachability of robot. Some limitations which are not imagined or calculated before the execution of path can be find out. With the help of this virtual environment, path followed by each and every joint while following helical trajectory or any other complex trajectory required can also be known well in advance and if in between during motion any constraint is violated, it can be rectified and corrected before implementing it on hardware which helps to avoid any damage and loss.

5.3 Structure Limitation

Every robot has pre-defined work envelope and reach capability which cannot be altered easily as its directly related with nature of joints and dimensional parameters of the robot structure. Then every joint is also restricted to move within its recommended range of motion which forms the joint space of robot. In this robot due to joint coupling, it is observed that even by actuating single joint, its adjacent joints also rotates to maintain the structure stability, therefore actual path followed by tool tip get deviated from the defined helical path.

Another factor called Load shaft precision also plays important role so as to follow the specified path. In rhino robot, each of the five axis are driven by DC servomotor that has an incremental encoder attached to the high speed shaft. The encoder resolves the high speed position to 60 degrees. Since each motor has a built in gear head with a turns ratio of 66.1:1 or 96:1, this results in a precision for each load shaft of 0.908 or 0.624 degrees per count. Finally, each joint has its own set of sprockets and chains which then determine the joint angle precision. The resulting precision in degrees per count for each joint is shown in the Table 5-1.

Motor	Joint	Precision
F	Base	0.2269
E	Shoulder	0.1135
D	Elbow	0.1135
C	Tool Pitch	0.1135
B	Tool Roll	0.3125

Table 5-1: Precise factor of each axis [18]

Although robotic manipulators are not nearly as accurate as their more expensive and more specialised hard automation counterparts, they do provide increased flexibility. It is one of the great challenges of robotics to make use of this increased flexibility to compensate for less than perfect accuracy. In particular, clever algorithms and control strategies are needed that

make use of external sensors to compensate for uncertainties in the environment and uncertainties in the exact position of robot itself.

Scope of Further Work

This work has demonstrated that flexibility and versatility in robots i.e. any desired trajectory can be traced using the inverse kinematics solution of the required path for any configuration using an open architecture approach, which provides custom programmed paths and the controller only transmits the coordinates to the motors. Based on this work it is concluded that the accuracy of the obtained path for operations like welding is within limits while for high precision works the selection of the required points on the desired path needs attention. Further, not just position but velocity needs to be controlled at every point, the computational load of this is easily taken by the computer away from the controller and the hard programming provided by manufacturer.

In real life implementation, in case of reconditioning of cylinder liners, amount of material to be filled inside the liner may not be uniform along the circular path, then consumption rate of electrode will not be uniform throughout. Therefore these factors can be considered further to optimise the given task.

The presented idea is not restricted to this specific application of reconditioning of a cylinder block of an IC engine. The given example is taken only to present the idea that a general purpose jointed arm configuration robot can be successfully programmed for tasks better suited for other configurations. Its applications can be extended for painting, cleaning and maintenance operations to be performed on cylinders.

6. APPENDIX

6.1 Rhino Robot Model Specifications

Model	XR-III Robot Arm
Applications	Education, training, research
Configuration	5 Axes plus gripper All axes completely independent All axes can be controlled simultaneously
Drives	Six stepper motors with integral gearboxes and incremental optical encoders
Controller	Mark-I Controller if XR-2 Motors are Version 4; Mark II Controller if Version 2 or 3
Payload	2.2 pounds (1 kilograms)
Speed (Axes)	15 inches per second at finger tips
Repeatability	0.157 inches at full extension
Weight	16 pounds (7.3 kg) without base; base is 10 pounds (4.5 kg)
Reach	22-1/2 inches from center of pivot
Work Envelope	Motor "F" Body Rotation - 360 degrees Motor "E" Shoulder Rotation - 250 degrees Motor "D" Elbow Rotation - 270 degrees Motor "C" Wrist Rotation - 270 degrees Motor "B" Gripper Rotation - +/- 7 revolutions
Standard Gripper	1 inch (25.4 mm) opening for position 1 2 inch (50.8 mm) opening for position 2

End Effector Options	Long finger attachment Triple finger attachment Narrow finger attachment Clam shell finger attachment Shovel attachment Rhino writer hand
Accessories	Rotary carousel Tilting rotary carousel X-Y table Belt conveyor, Slide base

THE CONTROLLER [16]

The rhino robot controller is a basic, 8 axis controller with the ability to interact with 8 TTL level input lines and to control 8 TTL level output lines. It also has the ability to control two auxiliary ports that provide 20 volts each at 1.5 amps. All input lines are pulled up to TTL high. Ground them for easy operation. No power is needed. Output lines provide TTL level signals at 100 milliamps. The auxiliary ports can be used to control two more motors or relays as needed by the experimenter. The Mark I controller integrates the power supplies, communication, microprocessor logic; teach pendant support, the input/output capability, all in one unit. The unit is ready to be connected to and used with the Rhino XR series or SCARA robot. The ease of use of an integrated system makes it easier for the students to learn & understand.

Figure 6.1: Rhino Robot Controller [16]

TEACH PENDANT

The teach pendant supports 32 keys and seven 7 segment display digits for visual feedback. It is easy to use. The teach pendant is standard equipment with each rhino robot controller.

Figure 6.2: Teach Pendant [16]

SPECIFICATIONS

Model:	Mark III controller and teach pendant
Part Number:	FG1006, includes teach pendant
Applications:	Education, industrial training, research
Configuration:	8 Port for optically encoded motors to operate the XR series and SCARA robots and accessories. Two ports for auxiliary motors.
Input/Output:	8 Input line pairs + 8 internal input lines 8 Output line pairs, TTL Full teach pendant support.

Microprocessor: 8 Bit 6502 main processor

Communications: RS-232C, full handshake direct pendant interface

Compatibility:	Can be run from any computer with an RS-232C interface.
Commands:	Over 16 kernel commands form a comprehensive control language.
Voltage:	Available in either 120 Volts or 240 Volts, 50 or 60 Hertz, single

phase.

Dimensions	18" wide x 13-1/2" deep x 6" high
Weight	24 lbs.

RHINO ROBOTICS PINOUTS [27]

The pin outs of the motor connector, as seen with the plug removed, are as follows:

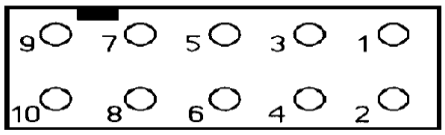


Figure 6.3: Pin Outs [27]

MARK III CONTROLLER

1. Logic Ground
2. Optic "B"
3. +5 Volts
4. Optic "A"
5. N/C
6. Limit Switch
7. Motor Common
8. Motor Common
9. Motor Power
10. Motor Power

RHINO ROBOTICS XR-3 MOTOR ENCODER OPTICS [17]

The Rhino Motor Encoder Disks have the following characteristics. The Small Motor Disks have six segments - three black and three silver. The Large Motor Disks have twelve segments - six black and six silver. The Optic Boards have two Optics (5 Volt Digital Square Wave) positioned 90 degree out of phase. This allows the board to have four different states. With the Optics positioned like this, it is possible to calculate velocity, acceleration and position.

The Mark III Controller counts three states per revolution on the Encoders for the Small Motors. The Mark III counts six states per revolution on the Encoders for the Large Motors. The Small Motors have a 96:1 Gear Ratio. Therefore, the Mark III Controller would read 288 counts per revolution of the Motor Output Shaft.

The Large Motors with a 65.5:1 Gear Ration generate 393 counts per revolution for the Mark III Controller.

MOTOR AND GEAR RATIOS [18]

The linkage ratio on each axis of the robot arm has further reduction.

Motor	Linkage	Total Reduction	Mark III Counts Per Degree	Mark III Theoretical Counts Per Revolution
B	Wrist Rotation	384:1	3.2	1152
C	Wrist Flex	524:1	8.7	3144
D	Elbow	524:1	8.7	3144
E	Shoulder	524:1	8.7	3144
F	Waist	262:1	4.4	1572

Table 6-1: Gear reduction specifications [18]

ROBOT MOTOR ADDITIONAL SPECIFICATIONS

	Continuous Armature Torque	Theoretical Peak Stall Torque	Reference Voltage	No Load Current	Theoretical Peak Stall Current	Maximum Gear Box Load	No Load Gear Box Speed
Small Motor	0.0089 N-M 1.3 oz-in	0.0357 N-M 5.1 oz-in	12-24 volts	0.11 A	1.40 A	0.71 N-M 100 oz-in	86.5 RPM
Large Motor 65.5	0.0226 N-M 3.2 oz-in	0.110 N-M 15.6 oz-in	12-20 volts	0.20 A	5.54 A	1.24 N-M 175 oz-in	85.3 RPM
SCARA "E" Large Motor 127.7	0.0226 N-M 3.2 oz-in	0.110 N-M 15.6 oz-in	12-20 volts	0.20 A	5.54 A	1.24 N-M 175 oz-in	43.8 RPM

Table 6-2: Electrical specification of motors [18]

RESOLUTION AT EACH AXIS

Axis	Motor	Resolution
Fingers	A	not applicable
Wrist Rotation	B	0.18 degrees theoretical
Wrist Flex	C	0.12 degrees theoretical
Forearm	D	0.12 degrees theoretical
Shoulder		
Waist	E	

Table 6-3: Motor resolution [18]

SPEED AT EACH AXIS

Axis	Speed(degrees/second)
Fingers	1 sec. to open and close fully
Wrist Rotation	32
Wrist Flex	
Elbow	45
Shoulder	30
Waist	

Table 6-4: Motor Speed [18]

ADDITIONAL TECHNICAL INFORMATION [18]

The approximate measurements of the XR-3 Robot are as follows:

- Bottom plate 0.750 to 1.000 inches thick.
- Base height 6 inches, (F motor) motor gear ratio either 66.1 to 1 or 65.5 to 1. Base pulley to motor pulley 120 to 18. Six encoder wheel slots.
- Shoulder axis height 3.75 inches from top of base. Shoulder axis to elbow axis 9 inches. (E motor) motor gear ratio either 66.1 to 1 or 65.5 to 1. Shoulder sprocket to motor sprocket ratio 72 to 9. Six encoder wheel slots. Shoulder angle set according to homing cam which actuates shoulder micro switch. The angle set at factory is approximately 45 to 55 degrees, however this should be measured.
- Elbow axis to wrist axis is 9 inches. (D motor) motor gear ratio either 66.1 to 1 or 65.5 to 1. Elbow sprocket to motor sprocket ratio 72 to 9. Six encoder wheel slots. Elbow angle set according to homing cam which actuates elbow micro switch. The angle set at factory is approximately horizontal, however this should be measured.
- Wrist flex is offset 0.375 inches from the center of the gripper open/close shaft.(C motor) motor gear ratio either 66.1 to 1 or 65.5 to 1. Wrist flex sprocket to motor sprocket ratio 72 to 9. Most common XR-2 wrist flex connection has a slip wire connection to the wrist axis. Six encoder wheel slots. Wrist flex angle is set according to

homing cam which actuates Wrist flex micro switch. The angle set at factory is approximately for a vertical hand which is 90 degrees from the horizontal elbow axis, however this should be measured as the slip wire allows for slip if the Robot crashes into something.

- The Wrist rotate and Gripper center are offset 0.375 inches from the wrist flex axis. Wrist rotate (B motor) motor gear ratio is probably 187 to 1 gear ratio. Newer motors have 96 to 1 gear ratios. Wrist rotate sprocket to motor sprocket is 64 to 16. Two encoder wheel slots. Wrist rotate angle is set according to homing cam which actuates Wrist rotate micro switch.
- The most common setting is for on of the finger tips to be at the front of the Robot so the finger links are parallel with the side of the hand. The other common setting is 90 degrees from that position.

SERVO

The Rhino uses a closed loop encoder system that works well when adjusted correctly. First is the encoder wheel. As the wheel turns it feeds data to the controller. From this data the controller determines if the correct position. Let's say the motor has turned when it was not supposed to. The controller would turn the motor on in the opposite direction to return it to the desired position This is fine expect the encoder now has gone to far in the other direction. Fig. shows basic control loop of servos. A feedback system that consists of a sensing element, an amplifier and a servomotor used in the automatic control of a mechanical device.

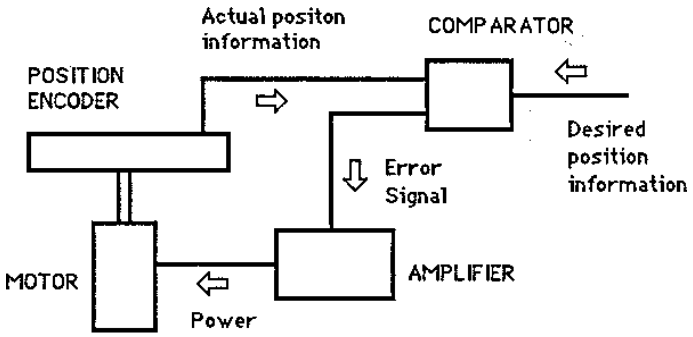


Figure 6.4: Servo diagram [5]

ENCODER [17]

Encoder is used to find the position of the servo motor. The encoders used on the Rhino system are incremental encoders. Each consists of an aluminum disk with light and dark bands placed radially on one side. Two reflective optical sensors are located to detect the different bands and placed so as to produce two signals (A and B) which are 90 degrees out of phase. The large motors (C-F) have six dark and six light bands per revolution; the remaining motors have 3 sets of bands.

When the A signal leads the B signal, the motor is moving in one direction and when the A signal trails the B signal, the motor is moving in the other direction. The logic in the controller is arranged to be able to make the necessary discrimination.

Given that there are two signals from the encoder, there are four states that the incremental encoder can provide. They are 00, 01, 11 and 10. Note that if these were interpreted as decimal values, the flow is from 0 to 1 to 3 to 2 and back to zero, not 1, 2, 3 and 4. Also note that it is not possible to go from 00 to 11 without going through one of the intermediate states and that it is not possible to go from 10 to 01 without going through an intermediate step. These are called forbidden states and if these changes occur, they indicate an error in operation. Fig. shows the transition of the encoder state.

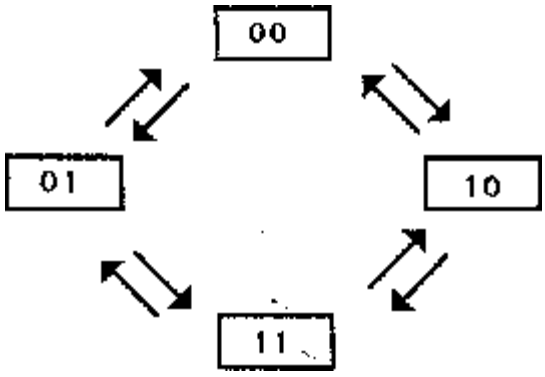


Figure 6.5: Encoder signal flow [17]

A counter can be set up using either 1,2 or 4 positions of the encoder. If we pick one position, we would increment or decrement the counters only when the states went all the way around the above diagram. This method is used in the Mark III controller. If we picked two positions, we would increment or decrement the counters whenever the state changed from one side of the diagram to the other side on the diagram. If we picked four positions, we would change the encoder count every time the state changed.

The controller uses eight registers in its memory as error registers for the motors. As long as a register is zero, the motor has no power applied to it. If an encoder turns, the register is added to or subtracted from as needed. As soon as the controller sees a number in the error register, it applies power to the motor in a way that will turn it in the direction that will decrement the register as the motor turns. This is the holding algorithm that maintains motor position at all times.

When the controller receives a START command from the host computer, it adds the given number to the motor error register. This has the effect of making the controller start the motor in a direction that will make that error register go to zero. Each cycle of the encoder changes the error register by one. When the error hoes to zero, the motor is turned off. If the motor overshoots, the power to the motor will be reversed automatically to bring the motor back to the zero position.

With the Rhino XR-3 robot, the large motors have 6 detectable state per motor revolution and the small motors have 3 states per revolution.

OPTICS

The optics system is the electronic circuitry that feeds; information back to me controller with regard to me position of the motor. The information needed by the controller is the position of the motor and the status of the micro switch. There are a few key assemblies used in the optics system to perform these functions. They follow, and each is very necessary to have the robot function correctly.

HOME POSITION

The Rhino system has two types of HOME positions. The HARD HOME is a mechanical home position which is determined by limit switches located on the waist, shoulder, elbow, wrist flex, and wrist rotate axes. The SOFT HOME is a software home and can be set at any position in the work envelope. Hard home must be performed before a new move sequence is taught and before a stored sequence is loaded and executed. The soft home position is a point in the work envelope selected by the programmer because it is an efficient starting point for the desired move sequence.

6.2 Introduction to Peripheral Interface Controller (PIC)

PIC [19] is a family of modified Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1650 originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to "Peripheral Interface Controller". PICs are popular with both industrial developers and hobbyists alike due to their low cost, wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, and serial programming (and re-programming with flash memory) capability.

Microchip announced on September 2011 the shipment of its ten billionth PIC processor.

Development

Since 1998, Microchip Technology continuously developed new high performance microcontrollers with new complex architecture and enhanced in-built peripherals. PIC microcontroller is based on Harvard architecture. At present PIC microcontrollers are widely used for industrial purpose due to its high performance ability at low power consumption. It is

also very famous among hobbyists due to moderate cost and easy availability of its supporting software and hardware tools like compilers, simulators, debuggers etc.

Core Architecture

The PIC architecture is characterized by its multiple attributes:

- Separate code and data spaces (Harvard architecture) for devices other than PIC32, which has a different architecture.
- A small number of fixed length instructions
- Most instructions are single cycle execution (2 clock cycles, or 4 clock cycles in 8-bit models), with one delay cycle on branches and skips
- One accumulator (W0), the use of which (as source operand) is implied (i.e. is not encoded in the opcode)
- All RAM locations function as registers as both source and/or destination of math and other functions.
- A hardware stack for storing return addresses
- A fairly small amount of addressable data space (typically 256 bytes), extended through banking.
- Data space mapped CPU, port, and peripheral registers
- The program counter is also mapped into the data space and writable (this is used to implement indirect jumps).

There is no distinction between memory space and register space because the RAM serves the job of both memory and registers, and the RAM is usually just referred to as the register file or simply as the registers.

ADVANTAGES

The PIC architectures have these advantages:

- Small instruction set to learn
- Reduced instruction set computing (RISC) architecture
- Built in oscillator with selectable speeds
- Easy entry level, in circuit programming plus in circuit debugging PIC Kit units available from Microchip.com for less than $50
- Inexpensive microcontrollers
- Wide range of interfaces including I²C, SPI, USB, USART, A/D, programmable comparators, PWM, LIN, CAN, PSP, and Ethernet.

LIMITATIONS

The PIC architectures have these limitations:

- One accumulator
- Register-bank switching is required to access the entire RAM of many devices
- Operations and registers are not orthogonal; some instructions can address RAM and/or immediate constants, while others can only use the accumulator

The following stack limitations have been addressed in the **PIC18** series, but still apply to earlier cores:

- The hardware call stack is not addressable, so pre-emptive task switching cannot be implemented
- Software-implemented stacks are not efficient, so it is difficult to generate re-entrant code and support local variables.

6.3 Why PIC Chosen for Servomotor Control

The PIC microcontroller makes an ideal choice [20] for an embedded DC Servomotor application. The PIC-micro family has many devices and options for the embedded designer to choose from. Furthermore, pin compatible devices are offered in the PIC16CXXX and PIC18CXXX device family, which makes it possible to use either device in the same hardware design. This gives the designer an easy migration path, depending on the features and performance required in the application. In particular, this servomotor has been implemented on both the PIC18C452 and PIC16F877 devices, and we'll look at the MCU resources required to support the servomotor application. With an understanding of the servomotor functions, you can start with the design shown here and implement your own custom DC servomotor application based on the PIC micro device that suits your needs.

The PIC micro MCU handles many functions in the servomotor application, such as:

• User control interface

• Measurement of motor position

• Computation of motion profile

• Computation of error signal and PID compensation algorithm

• Generation of motor drive signal

• Communication with non-volatile EEPROM memory.

SERVOMOTOR SOURCE CODE [21]

Two source code listings are provided with this application note. The source code will be written in miKroC PIC compiler and will operate on a PIC16XXXX. If the LEDs are omitted from the design, the code may also be compiled to operate on the PIC16XXXX or the PIC18XXXX, which are 28-pin devices with fewer I/O pins. The PIC18C452 application utilizes the MSSP peripheral to read and write data to a 24C01 serial EEPROM. The code will be written to utilize the on-chip data EEPROM to store profile data and PID gain values.

These routines may be omitted, if we wish to compile the source code for other devices in the PIC16CXXX family. Like the PIC18C452 source code, the LEDs may be removed for operation on a lower pin-count device.

Although the hardware-based solution for decoding the quadrature encoder signals requires minimal software overhead, it does require that two timers be used for each encoder. There are not enough timer resources available on the PIC18C452 device to permit two encoders to be decoded. However, it is possible to decode the encoder outputs directly in software. Furthermore, the two priority level interrupt structure of the PIC18CXXX architecture, provides an elegant way to handle the software decoding algorithm. Using priority interrupts, the servo calculations are performed in the low priority ISR and the decoding algorithm is performed in the high priority ISR. The high priority ISR is able to override a low priority interrupt that is in progress. This is important in this case, because the encoder pulses can have short durations and must be processed quickly.

PIC16F882/883/884/886/887 FEATURES [21]:

High-Performance RISC CPU:

• Only 35 Instructions to Learn:

- All single-cycle instructions except branches

• Operating Speed:

- DC – 20 MHz oscillator/clock input

- DC – 200 ns instruction cycle

• Interrupt Capability

• 8-Level Deep Hardware Stack

• Direct, Indirect and Relative Addressing modes.

Special Microcontroller Features:

• Precision Internal Oscillator:

- Factory calibrated to ±1%

- Software selectable frequency range of 8 MHz to 31 kHz

- Software tunable

- Two-Speed Start-up mode

- Crystal fail detect for critical applications

- Clock mode switching during operation for power savings

• Power-Saving Sleep mode

• Wide Operating Voltage Range (2.0V-5.5V)

• Industrial and Extended Temperature Range

• Power-on Reset (POR)

• Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)

• Brown-out Reset (BOR) with Software Control Option

• Enhanced Low-Current Watchdog Timer (WDT) with On-Chip Oscillator (software selectable nominal 268 seconds with full prescaler) with software enable.

• Multiplexed Master Clear with Pull-up/Input Pin

• Programmable Code Protection

• High Endurance Flash/EEPROM Cell:

- 100,000 write Flash endurance

- 1,000,000 write EEPROM endurance

- Flash/Data EEPROM retention: > 40 years

• Program Memory Read/Write during run time

• In-Circuit Debugger (on board)

- Compiler Optimized Architecture with Optional Extended Instruction Set
- 100,000 Erase/Write Cycle Enhanced Flash
- Program Memory Typical
- 1,000,000 Erase/Write Cycle Data EEPROM Memory Typical
- Flexible oscillator option
- Four Crystal modes, including High-Precision PLL for USB
- Two External Clock modes, Up to 48 MHz
- Internal Oscillator: 8 user-selectable frequencies, from 31 kHz to 8 MHz
- Dual Oscillator Options allow Microcontroller and USB module to Run at different Clock Speeds. Master and Slave Modes with I2C Address Mask.

Low-Power Features:

- Standby Current:- 50 nA @ 2.0V, typical
- Operating Current:-11 μA @ 32 kHz, 2.0V, typical-220 μA @ 4 MHz, 2.0V, typical
- Watchdog Timer Current:-1 μA @ 2.0V, typical

Peripheral Features:

• 24/35 I/O Pins with Individual Direction Control:

- High current source/sink for direct LED drive

- Interrupt-on-Change pin

- Individually programmable weak pull-ups

- Ultra Low-Power Wake-up (ULPWU)

• Analog Comparator Module with:

- Two analog comparators

- Programmable on-chip voltage reference

(CVREF) module (% of VDD)

- Fixed voltage reference (0.6V)

- Comparator inputs and outputs externally accessible

- SR Latch mode

- External Timer1 Gate (count enable)

• A/D Converter:

- 10-bit resolution and 11/14 channels

• Timer0: 8-bit Timer/Counter with 8-bit

Programmable Prescaler

• Enhanced Timer1:

- 16-bit timer/counter with prescaler

- External Gate Input mode

- Dedicated low-power 32 kHz oscillator

• Timer2: 8-bit Timer/Counter with 8-bit Period

Register, Prescaler and Postscaler

• Enhanced Capture, Compare, PWM+ Module:

- 16-bit Capture, max. resolution 12.5 ns

- Compare, max. resolution 200 ns

- 10-bit PWM with 1, 2 or 4 output channels.

- PWM output steering control

Capture, Compare, PWM Module:

- 16-bit Capture, max. resolution 12.5 ns

- 16-bit Compare, max. resolution 200 ns

- 10-bit PWM, max. frequency 20 kHz

Enhanced USART Module:

- Supports RS-485, RS-232, and LIN 2.0

- Auto-Baud Detect

- Auto-Wake-Up on Start bit

In-Circuit Serial ProgrammingTM (ICSPTM) via Two Pins, Master Synchronous Serial Port (MSSP) Module supporting 3-wire SPI (all 4 modes) and I2C™

PIC16F882/883/884/886/887 [21]

Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	ECCP / CCP	EUSART	MSSP	Comparators	Timers 8/16-bit
	Flash (words)	SRAM (bytes)	EEPROM (bytes)							
PIC16F88	2048	128	128	24	11	1/1	1	1	2	2/1
PIC16F88	4096	256	256	24	11	1/1	1	1	2	2/1
PIC16F88	4096	256	256	35	14	1/1	1	1	2	2/1
PIC16F88	8192	368	256	24	11	1/1	1	1	2	2/1
PIC16F88	8192	368	256	35	14	1/1	1	1	2	2/1

Table 6-5: PIC16F88X specifications [21]

Parameter Name	Value
Program Memory Type	Flash
Program Memory (KB)	14
CPU Speed (MIPS)	5
RAM Bytes	368
Data EEPROM (bytes)	256
Digital Communication Peripherals	1-A/E/USART, 1-MSSP(SPI/I2C)
Capture/Compare/PWM Peripherals	1 CCP, 1 ECCP
Timers	2 x 8-bit, 1 x 16-bit
ADC	14 ch, 10-bit
Comparators	2
Temperature Range (C)	-40 to 125
Operating Voltage Range (V)	2 to 5.5
Pin Count	40
Cap Touch Channels	11

Table 6-6: PIC features [19]

PIC16F88X MEMORY PROGRAMMING SPECIFICATION [23]

Programming The PIC16f88x Devices

The PIC16F88X can be programmed using the high-voltage In-Circuit Serial ProgrammingTM (ICSPTM) method or the low-voltage ICSP method. Both of these can be done with the device in the user's system. The low-voltage ICSP method is slightly different than the high-voltage method and these differences are noted where applicable. This programming specification applies to these devices in all package types.

Program/Verify Mode

The Program/Verify mode for the PIC16F88X devices allows programming of the user program memory, data memory, user ID locations and the Configuration Word.

Programming and verification can take place in any memory region, independent of the remaining regions. This allows independent programming of program and data memory regions.

PIN DESCRIPTIONS IN PROGRAM/VERIFY MODE

Pin Name	During Programming		
	Function	Pin Type	Pin Description
RB3	PGM	I	Low-voltage ICSPTM programming input if LVP Configuration bit equals '1'
RB6	ICSPCLK	I	Clock Input – Schmitt Trigger input
RB7	ICSPDAT	I/O	Data Input/Output – Schmitt Trigger input
MCLR	Program/Verify mode	p(1)	Program Mode Select
VDD	VDD	P	Power Supply
VSS	VSS	P	Ground

Table 6-7: Pin Description [21]

Legend: I = Input, O = Output, P = Power

Note 1: In the PIC16F88X, the programming high voltage is internally generated. To activate the Program/Verify mode, high voltage needs to be applied to MCLR input. Since the MCLR is used for a level source, MCLR does not draw any significant current.

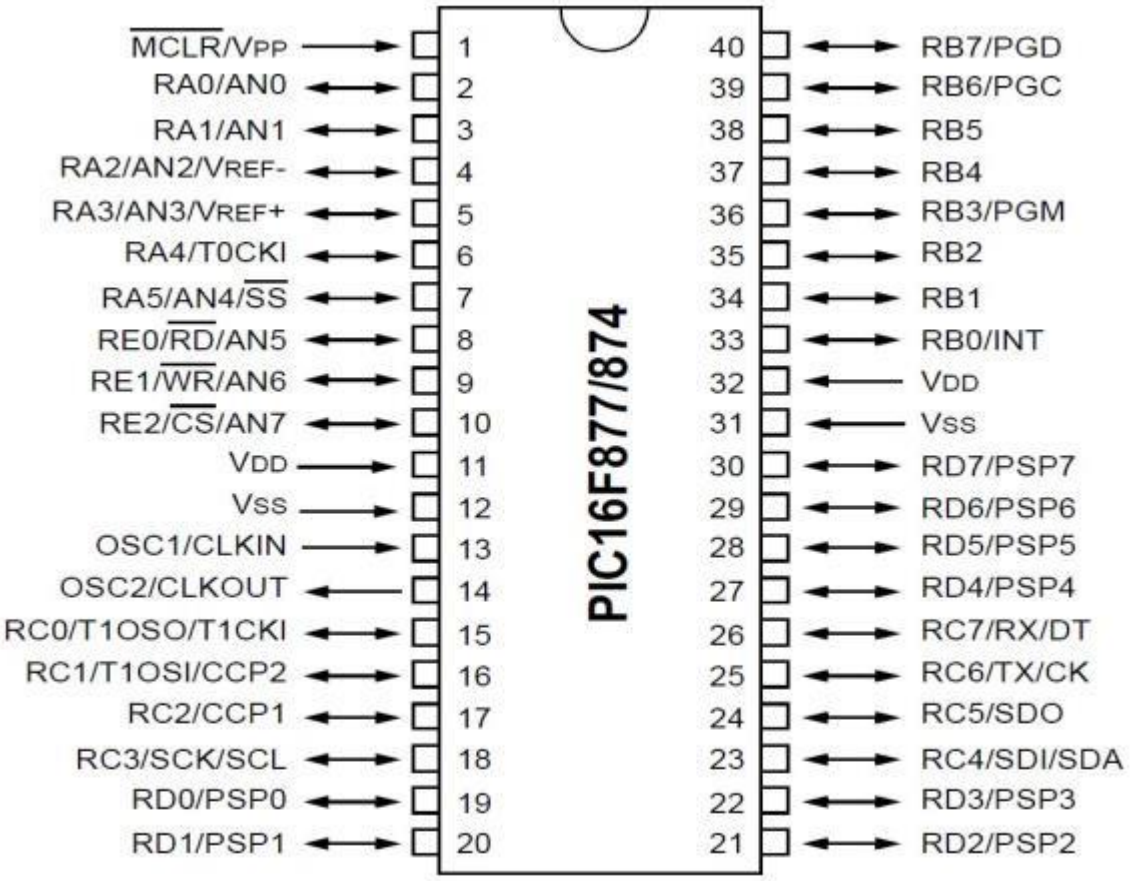


Figure 6.6: Controller Pins [21]

Low-Voltage ICSPTM Mode

The Low-Voltage ICSP Programming mode allows the PIC16F88X devices to be programmed using VDD only. However, when this mode is enabled by a Configuration bit (LVP), the PIC16F88X device dedicates RB3 to control entry/exit into Programming mode. When LVP bit is set to '1', the low-voltage ICSP programming entry is enabled. Since the LVP Configuration bit allows low-voltage ICSP programming entry in its erased state, an erased device will have the LVP bit enabled at the factory. While LVP is '1', RB3 is dedicated to low-voltage ICSP programming. Bring RB3 and then MCLR to VDD to enter Programming mode. All other specifications for high-voltage ICSP apply. To disable the Low-Voltage ICSP mode, the LVP bit must be programmed to '0'. This must be done while

entered in the High-Voltage Entry mode (LVP bit = '1'). RB3 is now a general purpose I/O pin.

Program/Erase Algorithms

The PIC16F88X devices' program memory may be written in three ways. The PIC16F882/883/884 uses one-word and four-word writes. The PIC16F886/887 uses one-word, four-word and eight-word writes. The four-word or eight-word algorithm is used to program the program memory only. The one-word algorithm can write any available memory location (i.e., program memory, configuration memory and data memory).

After writing the array, the PC may be reset and read back to verify the write. It is not possible to verify immediately following the write because the PC can only increment, not decrement.

A device Reset will clear the PC and set the address to '0'. The Increment Address command will increment the PC. The Load Configuration command will set the PC to 0x2000.

PICFlash [22]

PICFlash is a programmer and in-circuit debugger for PIC12F, PIC16F and PIC18F microcontrollers. PICFlash is mikroICD support - USB 2.0 programmer and mikroICD (In-Circuit Debugger). It is very fast, on-board USB 2.0 programmer with a simplified driver installation. mikroICD is a hardware tool designed for testing and debugging programs on PIC microcontrollers.

Figure 6.7: PICFlash Device [22]

Flash PIC microcontrollers allow erasing and reprogramming of the program memory in the microcontroller. Re-programmability offers a highly flexible solution to today's ever-changing market demands and can substantially reduce time to market. Users can program their systems late in the manufacturing process or update systems in the field. This allows easy code revisions, system parameterization or customer-specific options with no scrappage. Re-programmability also reduces the design verification cycle.

IN-CIRCUIT SERIAL PROGRAMMING™ (ICSP™)

Microchip's Flash and OTP PIC microcontrollers feature ICSP capability. ICSP allows the microcontroller to be programmed after being placed in a circuit board, offering tremendous flexibility, reduced development time, increased manufacturing efficiency and improved time to market. This popular technology enables reduced cost of field upgrades, system calibration during manufacturing and the addition of unique identification codes to the system. Requiring only two I/O pins for most devices, Microchip offers the most non-intrusive programming methodology in the industry.

Flash program is used to transfer a .hex file from a PC to the microcontroller memory by means of the appropriate hardware. Every flash program includes numerous options used for setting the microcontroller’s configuration bits. The PIC flash programmer is a great tool used for programming PIC microcontroller. Its unique design and ease of use make it a very

popular toll among beginners and professional users alike. The PIC flash programmer communicates to a PC through a USB cable which is also used for powering the programmer. In addition, it is a low power consumption device which makes it ideal for working with notebooks. In order to use this programmer, it is necessary to have the appropriate software mikroC installed on PC. The latest version of this software with updated list of supported microcontrollers can be downloaded free of charge from its website. Use any of its compilers to write a code as they provide an easy way of debugging/simulating the operation of the target device. The mikroICD debugger is part of the programmer to run that enables to run a program step by step while monitoring the state of all registers within the microcontroller. mikroICD is a highly effective tool for Real-Time debugging at hardware level. It is enabled to execute mikroC PRO, mikroPascal PRO and mikroBasic PRO programs on a host PIC microcontroller and monitor variable values, Special Function Registers (SFRs), RAM and EEPROM memory modules while the program is running. There is also an ultra fast USB 2.0 programmer for MCU programming that now supports more PIC microcontrollers.

PIC microcontrollers can be programmed at 5V and 12.5 V. If 5V is used, the RB3 pin is used for programming and thus cannot be used for other purposes. When the PICFlash performs programming at 12.5V, the RB3 pin is left free, as the 12V voltage is generated by a charge pump. Connection to a PC is established via the serial port, which means that the PICFlash programmer can operate on any PC.

System Specifications

Power supply: via a USB cable (5V DC)

Power consumption: 10 mA

Dimensions: 13 x 4 x 2.4 cm

Weight: 180g.

Connecting the Programmer

The PIC flash programmer is connected to the microcontroller via a flat cable ending with an IDC10 connector. The microcontroller may be soldered on the target device or plugged into the socket on the board intended for the microcontroller programming. In both cases, it is necessary to connect the microcontroller pins used for programming to a 2x5 connector. The

PIC flash programmer plastic case provides the IDC10 connector's pin out on the basis of which connection must be established between the microcontroller on the target device and the connector. The microcontroller is connected to the PICFlash programmer via 5 lines, two of which are +5V and GND. Unlike programmers whose operation is based on bootloaders and needs to put a part of their memory at a bootload program's disposal, PICFlash programs MCUs externally so that the entire memory is available to the programmer.

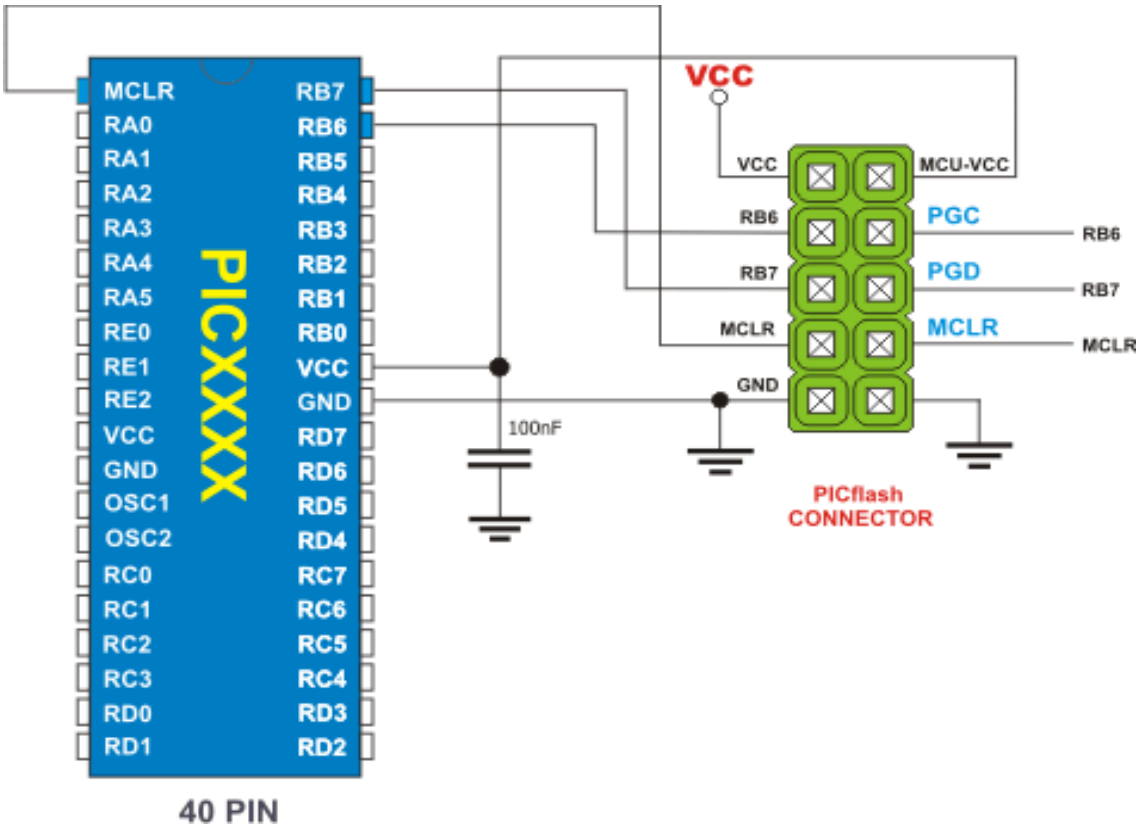


Figure 6.8: Controller Flash Connection [22]

Programming Robot

As the previously used language called RoboTalk is made for only rhino robots. It is the language with with predefined functions which are not easily understandable. Therefore with the implementation of new controller, the language used is C which is most common, easy to write and widely understandable.

The program is written in compiler known as mikroC commonly used for PIC controller. The complier deals with C language. The environment of the mikroC complier is shown in the Figure 6.9.

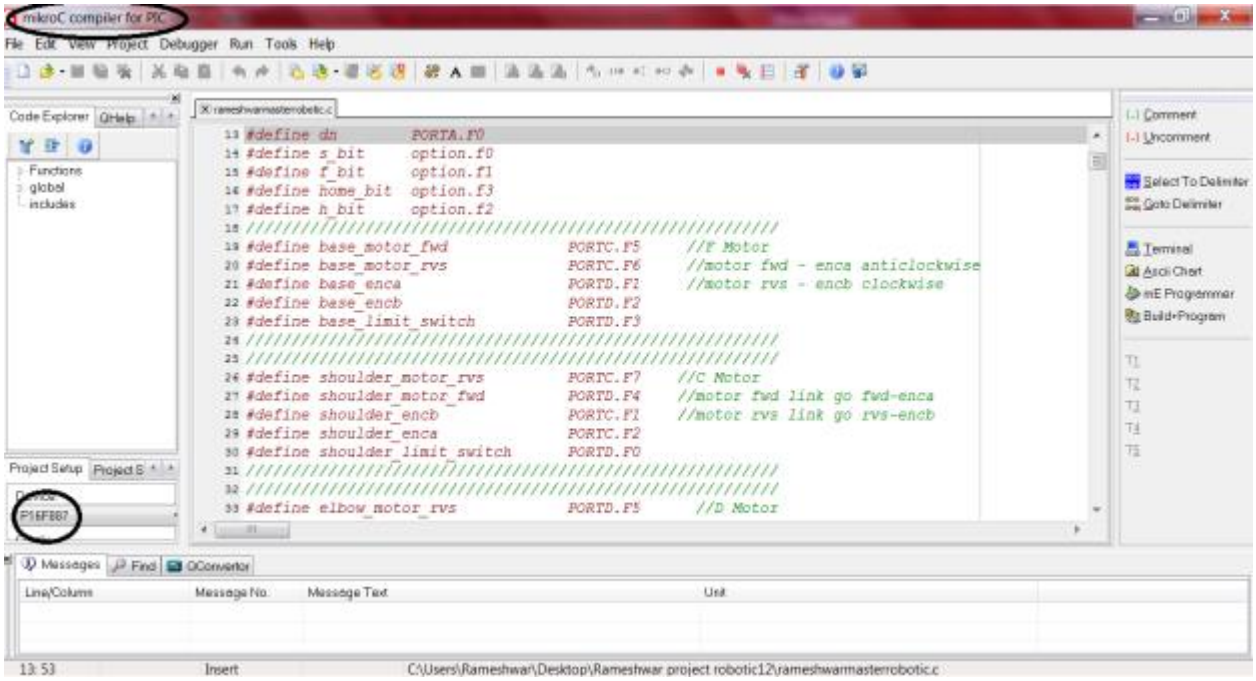


Figure 6.9: miKroC Environment

For every axis, specific formulae's are written to calculate respective joint angles for Rhino Robot. Once the program is fully defined in compiler, it is ready to burn onto the controller using FLASH device used for PIC controller. Once the program is loaded on controller, input is given to the required parameters with the help of buttons provided below LCD on the outer box of controller, these are variables in the program. The PICFlash working environment is shown in the Figure 6.10.

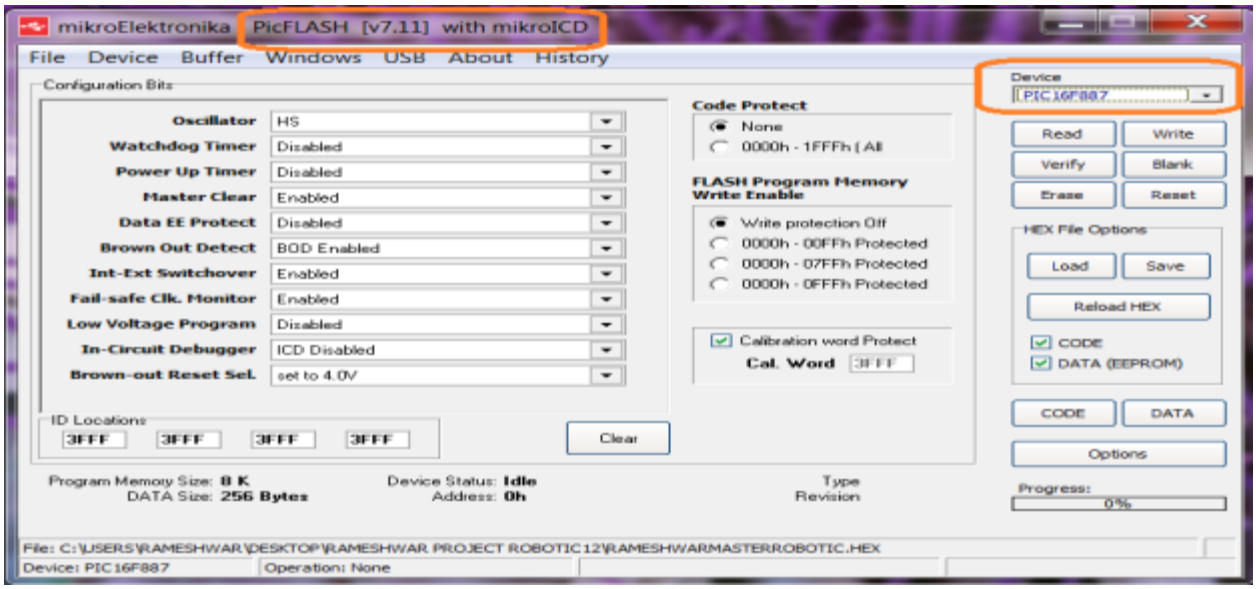


Figure 6.10: PICFlash Environment

After the execution of the calculation by controller, it sends further signals to the desired motors in a specified fashion so as to reach desired location. Depending upon the program written in compiler we can also control the path followed by the robot to reach desired location. Nature of path, Speed, Acceleration various parameters can be controlled as per our requirement.

To perform welding using robot on the inner periphery of cylinder, another algorithm for the helical path to be followed by the robot end effector is developed. It is also written using same compiler & then following same procedure to write it on controller. Then input variables are again entered using buttons provided below LCD, program burnt on the controller contains loops, once loop in the program started then after performing calculations, it keeps on giving the location point one after the other with a lag of just few nano seconds which becomes a continuous path control for robotic arm.

6.4 Programming a Microcontroller

Microcontrollers and humans communicate through the medium of the programming language called Assembly language [23]. The word Assembler itself does not have any deeper meaning, it corresponds to the names of other languages such as English or French. More precisely, assembly language is only a passing solution. In order that the microcontroller can understand a program written in assembly language, it must be compiled into a language of zeros and ones. Assembly language and Assembler do not have the same meaning. The first one refers to the set of rules used for writing program for the microcontroller, while the later refers to a program on a personal computer used to translate assembly language statements into the language of zeros and ones. A compiled program is also called Machine Code. A "Program" is a data file stored on a computer hard disc (or in memory of the microcontroller, if loaded) and written according to the rules of assembly or some other programming language. Assembly language is understandable for humans because it consists of meaningful words and symbols of the alphabet. Let us take, for example the command "RETURN" which is, as its name indicates, used to return the microcontroller from a subroutine. In machine code, the same command is represented by a 14-bit array of zeros and ones understandable by the microcontroller. All assembly language commands are similarly compiled into the corresponding array of zeros and ones. A data file used for storing compiled program is called an "executive file", i.e. "HEX data file". The name comes from the hexadecimal presentation of a data file and has a suffix of "hex" as well, for example "probe.hex". After has been generated, the data file is loaded into the microcontroller using a programmer. Assembly language programs may be written in any program for text processing (editor) able to create ASCII data files on a hard disc or in a specialized work environment.

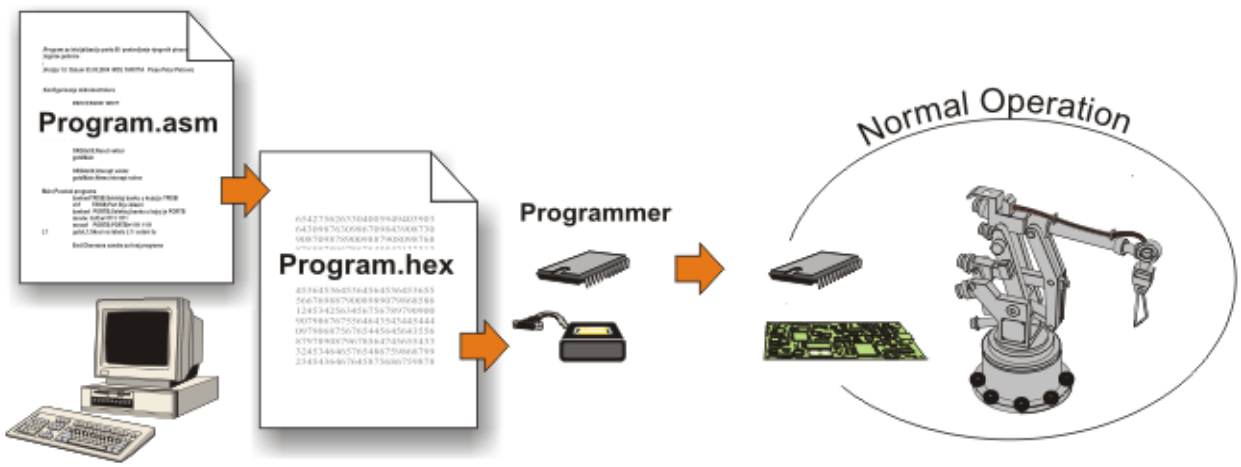


Figure 6.11 [23]

ELEMENTS OF ASSEMBLY LANGUAGE

A program written in assembly language consists of several elements being differently interpreted while compiling the program into an executable data file. The use of these elements requires strict rules and it is necessary to pay special attention to them during program writing in order to avoid errors.

ASSEMBLY LANGUAGE SYNTAX

As mentioned, it is necessary to observe some specific rules in order to enable the process of compiling into executive HEX code to run without errors. Compulsory rules explaining how sequences of expressions are put together to form the statements that make up an assembly language program are called syntax. There are only several of them:

- Every program line may consist of a maximum of 255 characters;
- Every program line that is to be compiled must start with a symbol, label, mnemonics or directive;
- Text following the mark ";" in a program line represents a comment which is ignored by the assembler (not compiled); and
- All the elements of one program line (labels, instructions etc.) must be separated by at

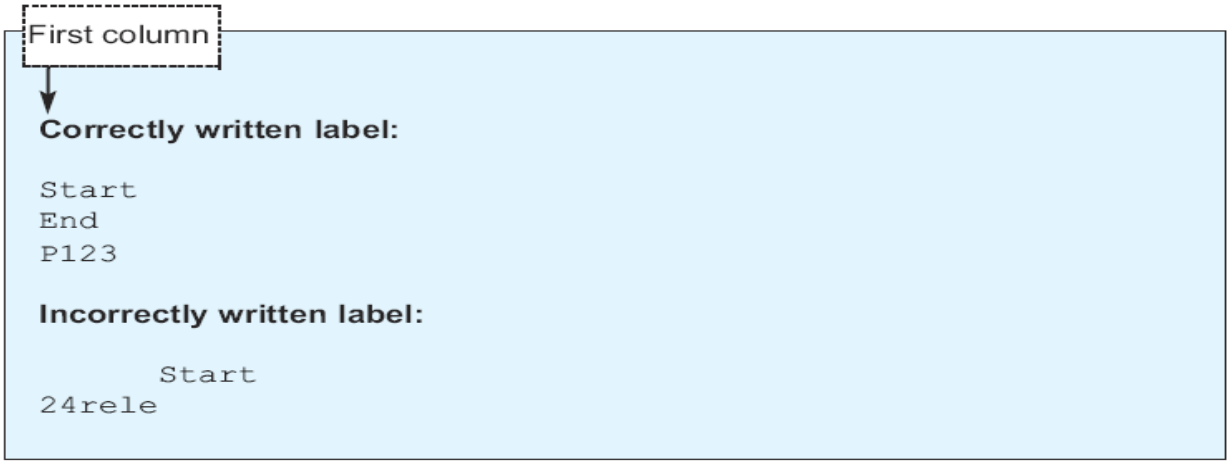
least one space character. For the sake of better clearness, a push-button TAB is commonly used instead of it, so that it is easy to delimit columns with labels, directives etc. in a program.

LABELS

A label represents a textual version of some address in ROM or RAM memory. Each label has to start in the first column with a letter of alphabet or "_" and may consist of maximum of 32 characters. Besides, it is easily used:

- It is sufficient to enter the name of a label instead of a 16-bit address in instruction which calls some subroutine or a jump. The label with the same name should also be written at the beginning of a program line in which a subroutine starts or where a jump should be executed. As a general rule, labels have easily recognizable names.

During program compiling, the assembler will automatically replace the labels by the corresponding addresses.



COMMENTS

A comment is often an explanatory text written by the programmer in order to make a program clearer and easier to understand. It is not necessary to comment every line. When three or four lines of code work together to accomplish some higher level task, it is better to have a single higher level comment for the group of lines. Therefore, it is added if needed and has to start with ";". Comments added to assembly source code are not compiled into machine code.

INSTRUCTIONS

Instructions are defined for each microcontroller family by the manufacturer. Therefore, it is up to the user to follow the rules of their usage. The way of writing instructions is also called instruction syntax. The instructions "movlp" and "gotto", in the following example, are recognized by the PIC16F887 microcontroller as an error since they are not correctly written.

Correctly written commands:

```
movlw H' FF'
goto Start
```

Incorrectly written commands:

```
movlp H' FF'
gotto Start
```

OPERANDS

An operand is a value (an argument) upon which the instruction, named by mnemonic, operates. The operands may be a register, a variable, a literal constant, a label or a memory address.

Using operand :

```
movlw H' 01F'
movwf LEVEL
```

operand as a variable LEVEL stored in the microcontroller memory

operand as a constant

END directive

Each program must be ended by using this directive. Once a program encounters this directive, the assembler immediately stops compiling. For example:

```
...

END ;End of program
```

\$INCLUDE directive

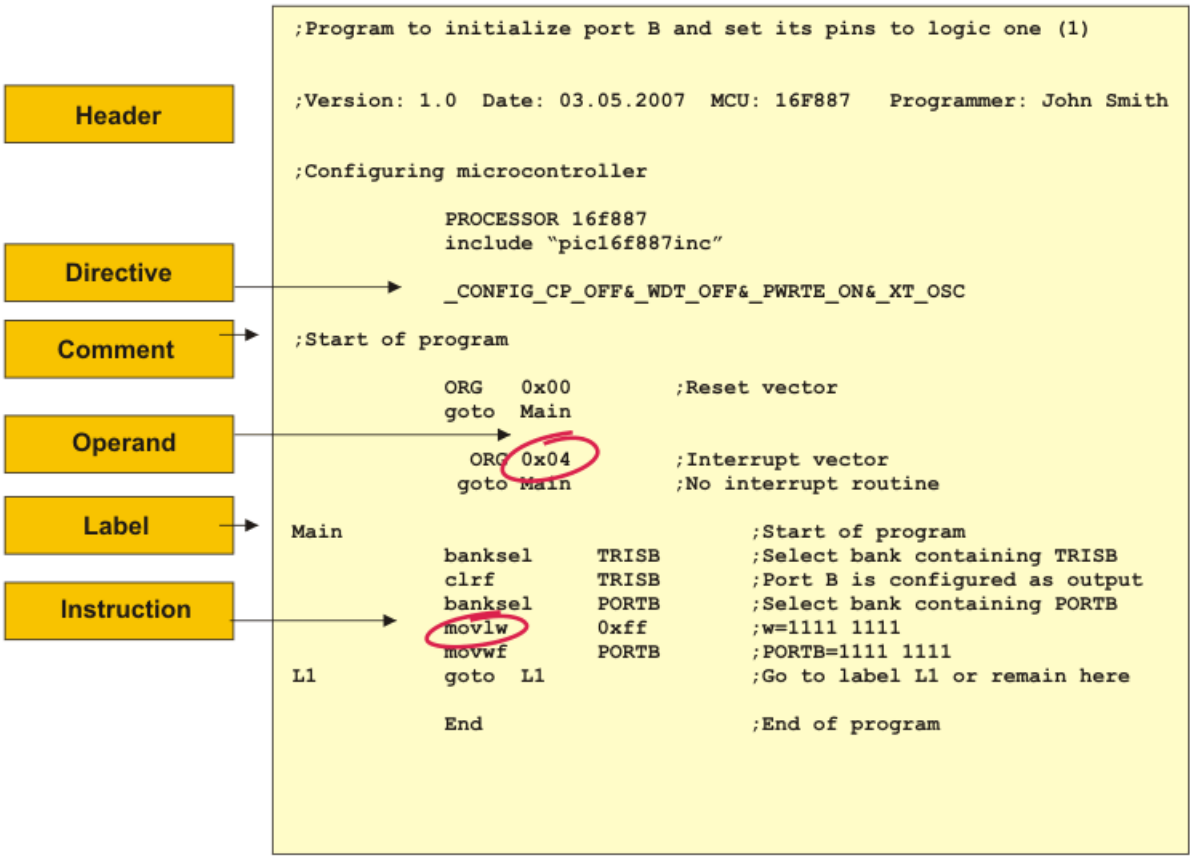
The name of this directive fully indicates its purpose. During compiling, it enables the assembler to use data contained in another file on a computer hard disc. For example:

```
.

#include <p16f887.inc>
```

6.5 Example of How to Write a Program

The following example illustrates what a simple program written in assembly language looks like.



Apart from the regular rules of assembly language, there are also some unwritten rules which should be observed during program writing. One of them is to write in a few words at the beginning of a program what the program's name is, what it is used for, version, release date, type of the microcontroller it is written for and the name of the programmer. Since this information is not of importance for the assembler, it is written as a comment which always starts with semicolon ';' and can be written in a new line or immediately after a command.

After writing this general comment, it is time to select the microcontroller by using directive PROCESSOR. This directive is followed by another one used to include all the definitions of the PIC16F887 microcontroller's internal registers in the program. These definitions are

nothing but the ability to address port B and other registers as PORTB instead of 06h, which makes the program clearer and more legible.

In order that the microcontroller will operate properly, a several parameters such as the type of oscillator, state of the watch-dog and internal reset circuit must be defined. It is done by utilizing the following directive:

```
CONFIG _CP_OFF&_WDT_OFF&PWRTE_ON&XT_OSC
```

When all necessary elements are defined, the process of program writing can start. First and foremost, it is necessary to specify the address from which the microcontroller starts when the power goes on (org 0x00) as well as the address from which the program proceeds with execution if an interrupt occurs (org 0x04). Since this program is very simple, it is enough to use command "goto Main" in order to direct the microcontroller to the beginning of the program. The next command selects memory bank 1 in order to enable access to the TRISB register to configure port B as output (banksel TRISB).

The main program ends by selecting memory bank 0 and setting all port B pins to logic one (1) (movlw 0xFF, movwf PORTB).

It is necessary to create a loop to keep program from "getting lost" in case an error occurs. For this purpose, there is an endless loop executed all the time while the microcontroller is switched on."end" is required at the end of every program to inform the assembler that there are no more commands to be compiled.

Examples

The purpose of this chapter is to provide basic information about microcontrollers that one needs to know in order to be able to use them successfully in practice. This chapter, therefore, does not contain any super interesting program or device schematic with amazing solutions. Instead, the following examples are more proof that program writing is neither a privilege nor a talent issue but the ability of simply putting puzzle pieces together using directives. Rest assured that design and development of devices mainly consists of the following method

"test-correct-repeat". Of course, the more you are in it the more complicated it becomes since the puzzle pieces are put together by both children and first-class architects

BASIC CONNECTING

As seen in the figure below, in order to enable the microcontroller to operate properly it is necessary to provide:

- Power Supply
- Reset Signal
- Clock Signal.

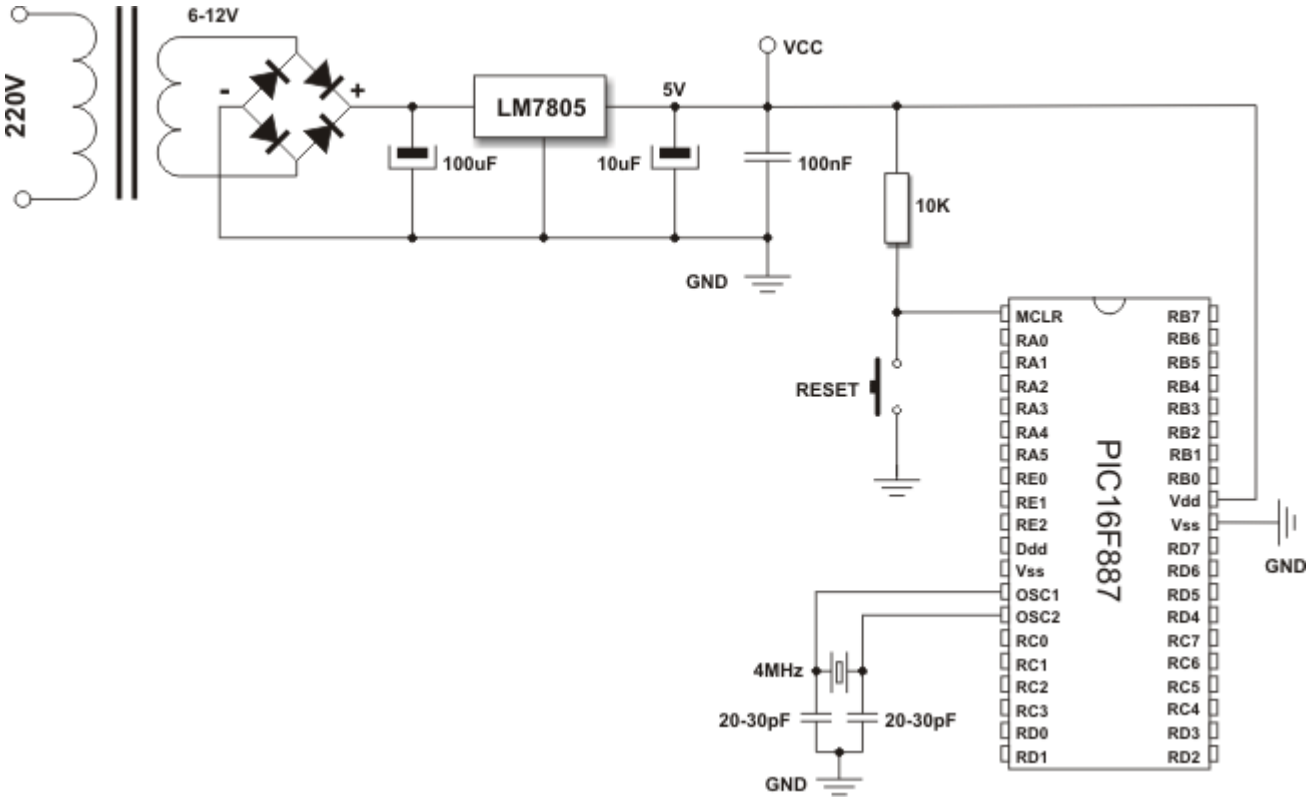


Figure 6.12: Circuit Diagram of PIC [23]

Clearly, it is about simple circuits, but it does not have to always be like that. If the target device is used for controlling expensive machines or maintaining vital functions, everything gets more and more complicated! However, this solution is sufficient for the time being.

POWER SUPPLY

Even though the PIC16F887 can operate at different supply voltages, why to test "Murphy's low"?! A 5DC power supply is shown above. The circuit, uses a cheap integrated three-terminal positive regulator, LM7805, provides high-quality voltage stability and quite enough current to enable microcontroller and peripheral electronics to operate normally (enough in this case means 1Amp).

RESET SIGNAL

In order that the microcontroller can operate properly, a logic one (VCC) must be applied on the reset pin it explains the connection pin-resistor 10K-VCC. The push-button connecting the reset pin MCLR to GND is not necessary. However, it is almost always provided because it enables the microcontroller safe return to normal operating conditions if something goes wrong. By pushing this button, 0V is brought to the pin, the microcontroller is reset and program execution starts from the beginning. The 10K resistor is there to allow 0V to be applied to the MCLR pin, via the push-button, without shorting the 5VDC rail to ground.

CLOCK SIGNAL

Even though the microcontroller has a built in oscillator, it cannot operate without external components which stabilize its operation and determine its frequency (operating speed of the microcontroller). Depending on which elements are in use as well as their frequencies, the oscillator can be run in four different modes:

- LP - Low Power Crystal
- XT - Crystal / Resonator
- HS - High speed Crystal / Resonator
- RC - Resistor / Capacitor.

Why are these modes so important? Owing to the fact that it is almost impossible to make a stable oscillator which operates over a wide frequency range, the microcontroller must know which crystal is connected in order that it can adjust the operation of its internal electronics to it. This is why all programs used for chip loading contains an option for oscillator mode selection. Refer Figure 6.13.

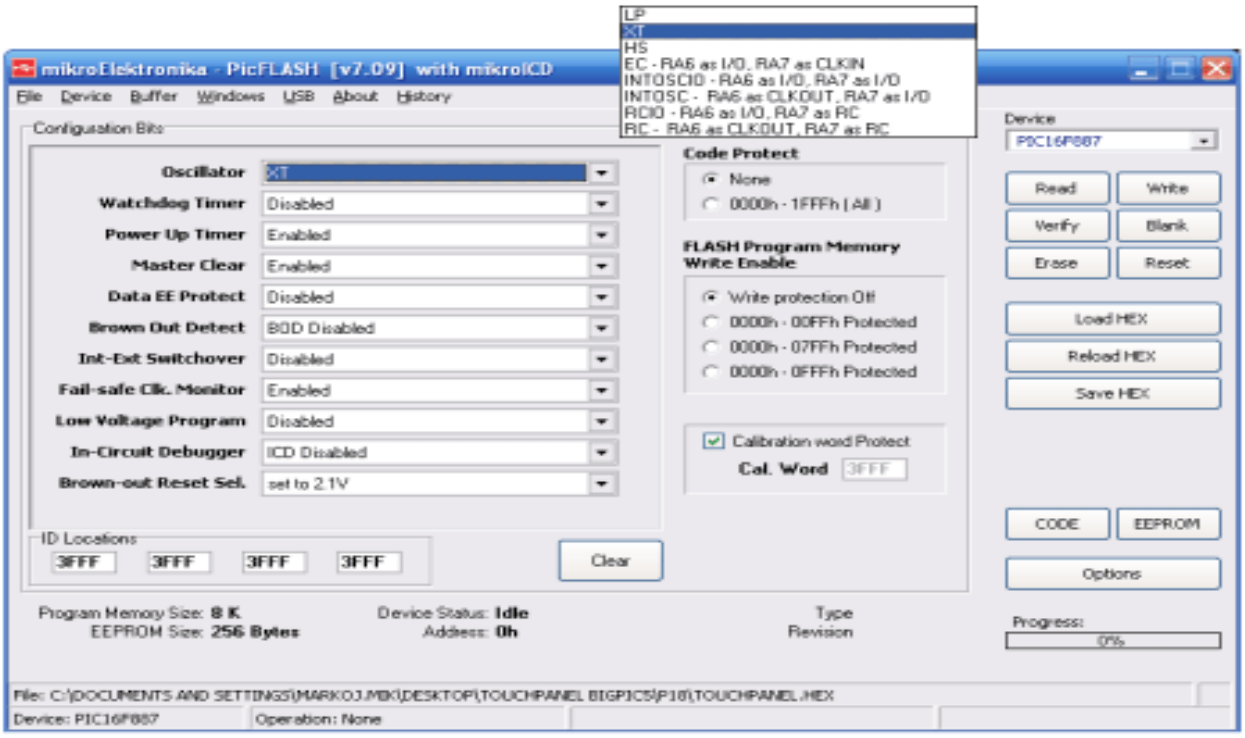


Figure 6.13: mikroC environment

RELAY

A relay is an electrical switch that opens and closes under the control of another electrical circuit. It is therefore connected to output pins of the microcontroller and used to turn on/off high-power devices such as motors, transformers, heaters, bulbs, etc. These devices are almost always placed away from the boards sensitive components. There are various types of relays, but all of them operate in the same way. When a current flows through the coil, the relay is operated by an electromagnet to open or close one or many sets of contacts. Similar to opto couplers, there is no galvanic connection (electrical contact) between input and output circuits. Relays usually demand both higher voltage and current to start operation but there are

also miniature ones that can be activated by a low current directly obtained from a microcontroller pin. The Figure 6.14 shows the most commonly used solution:

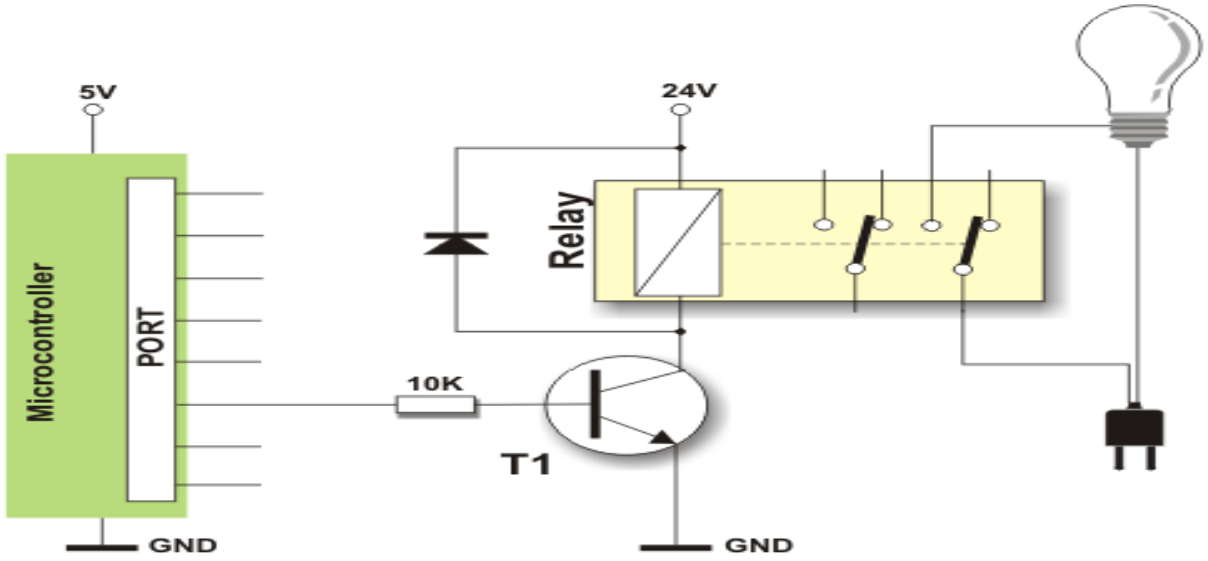


Figure 6.14: Circuit Layout [23]

LED DIODES

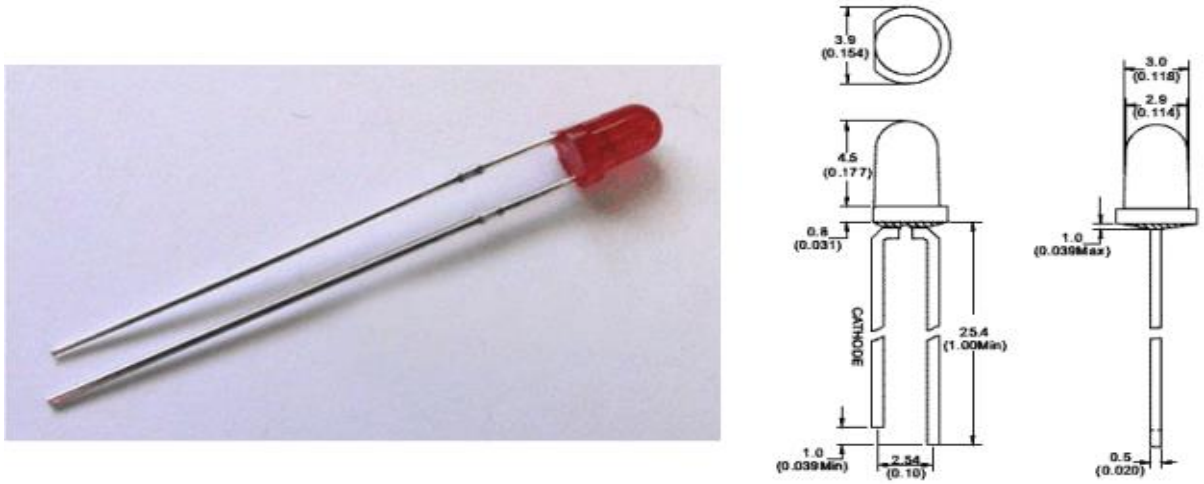


Figure 6.15: LED Diagram [23]

Quick burning

Like any other diode, LEDs have two ends an anode and a cathode. Connect it properly to a power supply voltage. The diode will happily emit light. Turn it upside down and apply the same power supply voltage (even for a moment). It will not emit light.

Slow burning

There is a nominal, i.e. maximum current determined for every LED which should not be exceeded. If it happens, the diode will emit more intensive light, but not for a long time.

Something to remember

Similar to the previous example, all you need to do is to discard a current limiting resistor shown below. Depending on power supply voltage, the effect might be spectacular!

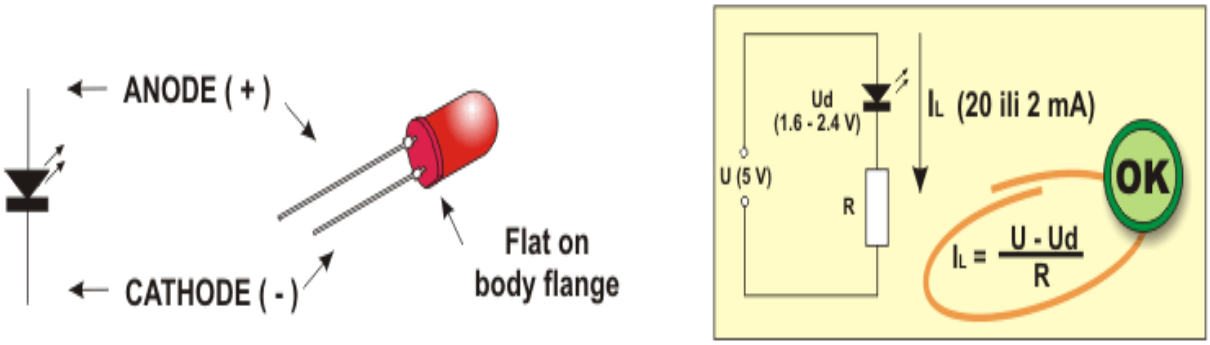


Figure 6.16: LED details [23]

LED DISPLAY

Basically, LED display is nothing more than several LEDs moulded in the same plastic case. There are many types of displays composed of several dozens of built in diodes which can display different symbols. The most commonly used is so called 7-segment display. It is composed of 8 LEDs- 7 segments are arranged as a rectangle for symbol displaying and there is an additional segment for decimal point displaying. In order to simplify connection, anodes or cathodes of all diodes are connected to the common pin so that there are common anode

displays and common cathode displays, respectively. Segments are marked with the letters from a to g, plus dp, as shown in Figure 6.17. On connecting, each diode is treated separately, which means that each must have its own current limiting resistor.

Figure 6.17: Pin Sequence [23]

Here are a few important things that one should pay attention to when buying LED displays:

- Depending on whether anodes or cathodes are connected to the common pin, there are common anode displays and common cathode displays. The Figure 6.17 above shows a common anode display. Looking at physical features, there is no difference between these displays at all so it is recommended to check carefully prior installation which of them is in use.
- For each microcontroller pin, there is a maximum current limitation it can receive or give. Because of this, if several displays are connected to the microcontroller it is recommended to use so called Low current LEDs using only 2mA for operation; and
- Display segments are usually marked with the letters from a to g, but there is no fast rule indicating to which micro controller pins they should be connected. For this reason it is very important to check connecting prior to commencing program writing or designing a device.

Displays connected to the microcontroller usually occupy a large number of valuable I/O pins, which can be a big problem especially when it is needed to display multi-digital numbers. The

problem is more than obvious if, for example, it is needed to display two 6-digit numbers (a simple calculation shows that 96 output pins are needed in this case)! This problem has a solution called MULTIPLEXING.

Here is how an optical illusion based on the same operating principle as a film camera is made. Only one digit at a time is active, but they change their state so quickly that one gets impression that all digits of a number are active simultaneously.

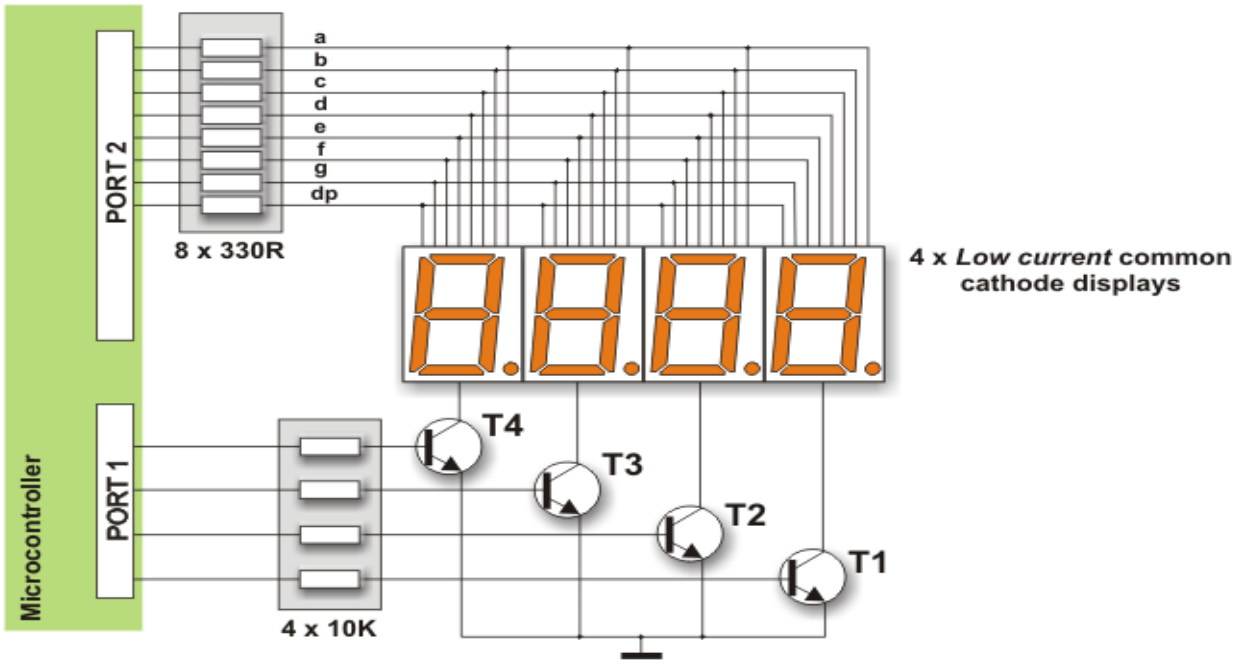


Figure 6.18: Pin Lay out [23]

Here is an explanation on the Figure 6.18. First a byte representing units is applied on a microcontroller port and a transistor T1 is activated simultaneously. After a while, the transistor T1 is turned off, a byte representing tens is applied on a port and transistor T2 is activated. This process is being cyclically repeated at high speed for all digits and corresponding transistors.

A disappointing fact which indicates that the microcontroller is just a kind of miniature computer designed to understand only the language of zeros and ones is fully expressed when displaying any digit. Namely, the microcontroller does not know what units, tens or hundreds

are, nor what ten digits we are used to look like. Therefore, each number to be displayed must go through the following procedure:

First of all, in a particular subroutine a multi-digital number must be split into units, tens etc. Then, these must be stored in special bytes each. Digits get recognizable format by performing "masking". In other words, a binary format of each digit is replaced by a different combination of bits using a simple subroutine. For example, the digit 8 (0000 1000) is replaced by binary number 0111 1111 in order to activate all LEDs displaying digit 8. The only diode remaining inactive in this case is reserved for the decimal point.

If a microcontroller port is connected to the display in a way that bit 0 activates segment "a", bit 1 activates segment "b", bit 2 segment "c" etc., then the table below shows the mask for each digit.

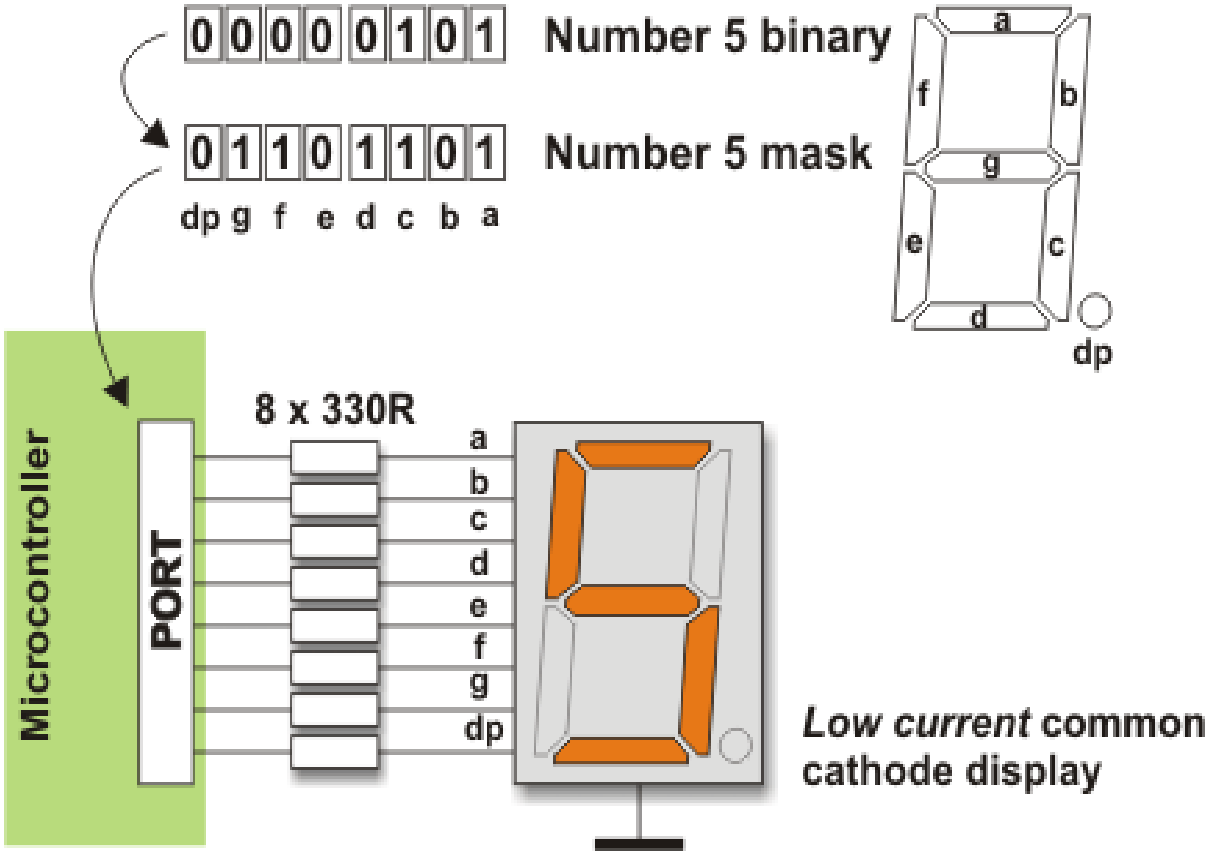


Figure 6.19: Port Numbers [23]

DIGITS TO DISPLAY	DISPLAY SEGMENTS							
	dp	a	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
1	0	0	1	1	0	0	0	0
2	0	1	1	0	1	1	0	1
3	0	1	1	1	1	0	0	1
4	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1
6	0	1	0	1	1	1	1	1
7	0	1	1	1	0	0	0	0
8	0	1	1	1	1	1	1	1
9	0	1	1	1	1	0	1	1

Table 6-8: Port Positions

In addition to digits from 0 to 9, there are some letters- A, C, E, J, F, U, H, L, b, c, d, o, r, t- that can be also displayed by means of the appropriate masking.

In the event that the common anode displays are used, all ones contained in the previous table should be replaced by zeros and vice versa. Additionally, NPN transistors should be used as drivers as well.

LCD DISPLAY

This component is specifically manufactured to be used with microcontrollers, which means that it cannot be activated by standard IC circuits. It is used for displaying different messages on a miniature liquid crystal display. The model described here is for its low price and great capabilities most frequently used in practice. It is based on the HD44780 microcontroller (Hitachi) and can display messages in two lines with 16 characters each. It displays all the letters of alphabet, Greek letters, punctuation marks, mathematical symbols etc. In addition, it

is possible to display symbols made up by the user. Other useful features include automatic message shift (left and right), cursor appearance, LED backlight etc.

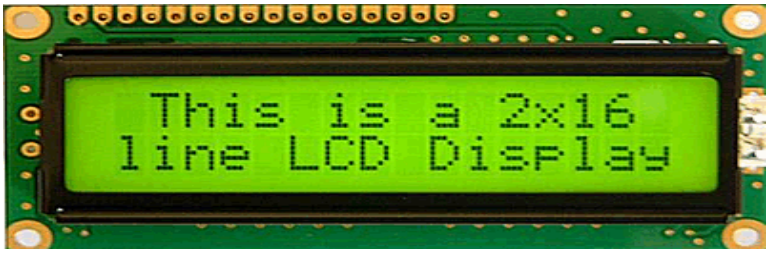


Figure 6.20: LCD Display

Along one side of a small printed board there are pins used for connecting to the microcontroller. There are in total of 14 pins marked with numbers. Their function is described in the Table 6-9:

FUNCTION	PIN NUMBER	NAME	LOGIC STATE	DESCRIPTION
Ground	1	Vss	-	0V
Power supply	2	Vdd	-	+5V
Contrast	3	Vee	-	0 - Vdd
Control of operating	4	RS	0 1	D0 – D7 are interpreted as commands D0 – D7 are interpreted as data
	5	R/W	0 1	Write data (from controller to LCD) Read data (from LCD to controller)
	6	E	0 1 From 1 to 0	Access to LCD disabled Normal operating Data/commands are transferred to LCD
Data / commands	7	D0	0/1	Bit 0 LSB
	8	D1	0/1	Bit 1
	9	D2	0/1	Bit 2
	10	D3	0/1	Bit 3

	11	D4	0/1	Bit 4
	12	D5	0/1	Bit 5
	13	D6	0/1	Bit 6
	14	D7	0/1	Bit 7 MSB

Table 6-9: Pin Description [23]

Development Systems

6.6 How to Start Working

A microcontroller is a good-natured "genie in the bottle" and no extra knowledge is required to use it.

In order to create a device controlled by the microcontroller, it is necessary to provide the simplest PC, program for compiling and simple device to transfer that code from PC to chip itself.

Even though this process is quite logical, there are often some queries, not because it is complicated, but for numerous variations.

WRITING PROGRAM IN ASSEMBLY LANGUAGE

In order to write a program for the microcontroller, a specialized program in the Windows environment may be used. Any program for text processing can be used for this purpose. The point is to write all instructions in such an order they should be executed by the microcontroller, observe the rules of assembly language and write instructions exactly as they are defined. In other words, you just have to follow the program idea! That's all! When using custom software, there are numerous tools which are also installed to aid in the development process. One such tool is the Simulator. This enables the user to test the code prior to burning it to the MCU.

```
oop      button PORTA,0,0,Increment
          button PORTA,1,0,Decrement
          goto Loop
Increment incf cnt,f
          movf cnt,w
          movwf PORTB
          goto Loop
Decrement decf cnt,f
          movf cnt,w
          movwf PORTB
```

To enable the compiler to perform its task successfully, it is necessary that a document containing this program has the extension, .asm in its name, for example: Program.asm

COMPILING PROGRAM

The microcontroller does not understand assembly language as such. This is why it is necessary to compile the program into machine language. It is more than simple when using a specialized program (MPLAB) because a compiler is part of the software! Just one click on the appropriate icon solves the problem and a new document with .hex extension pops out. It is actually the same program, but compiled into computer language which the microcontroller perfectly understands. Such documentation is commonly named "hex code" and seemingly represents a meaningless sequence of numbers in hexadecimal numerical system.

In case other software for program writing in assembly language is used, special software for compiling the program must be installed and used as follows: set up the compiler, open the

document with .asm extension and compile. The result is the same- a new document with .hex extension. The only problem you have now is that it is stored in your PC.

PROGRAMMING A MICROCONTROLLER

To enable "hex code" transmission to the microcontroller it is necessary to provide a cable for serial communication and a special device called programmer with appropriate software. There are several ways to do it.

A lot of programs and electronic circuits having this purpose can be found on the Internet. Do as follows: open hex code document, set a few parameters and click the icon for compiling. After a while, a sequence of zeros and ones is to be programmed into the microcontroller through the serial connection cable and programmer hardware. There is nothing else to be done except for placing the programmed chip into the target device. In case it is necessary to make some changes in the program, the previous procedure may be repeated an unlimited number of times.

The program is written and successfully compiled. All that's left is to dump the program to the microcontroller. For this purpose it is necessary to have software that takes the written and compiled program and passes it into the microcontroller (PIC Flash for example). Start up this program.

The settings are simple and there is no need for additional explanations (the type of the microcontroller, frequency and clock oscillator etc.)

- Connect the PC and programmer via a USB cable;
- Load the HEX code using command: File -> Load HEX; and
- Click the "Write" push-button and wait.

The microcontroller is programmed and everything is ready for operation. If working is not satisfied, make some changes in the program and repeat the procedure. During working of PIC Flash, a window appears on the screen as shown in the Figure 6.21

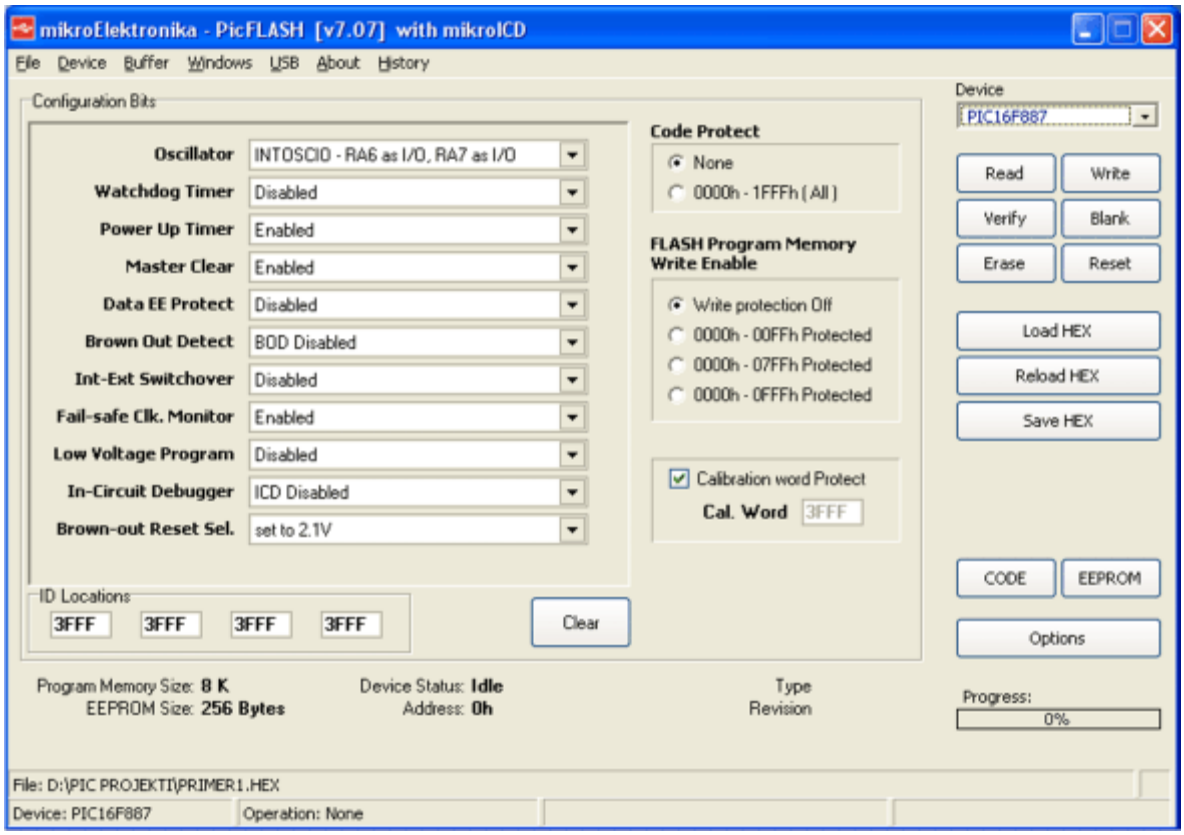


Figure 6.21: MikroC window

Development Systems

A device, which in testing program phase, can simulate any environment, is called a development system. Apart from the programmer, the power supply unit and the microcontroller™ socket, the development system contains elements for input pin activation and output pin monitoring. The simplest version has every pin connected to one push-button and one LED as well. A high quality version has LED displays, LCD displays, temperature sensors and all other elements which the target device can be supplied with. These peripherals could be connected to the MCU via miniature jumpers. In this way, the whole program may be tested in practice, during its development stage, because the microcontroller does not know, or care, whether its input is activated by a push-button or a sensor built in a real machine.

Robotalk Language [18]

RoboTalk serves as a specification for robust communication of robot configuration information and motion commands between a high-level control application and the robot hardware. Previous tele operation systems have been designed to cope with the problem of scheduling, and sending motion commands to a robot over a communication link. The Robonaut [6] architecture allows high-level commands to be converted by modules called sub autonomies into low-level motor commands. The Athena software development model [7] for Mars rovers features command sequencing to schedule and execute commands sent from the control application to the rover. RoboTalk allows control applications to be written in a manner that can be easily reused or reconfigured at run time for changing robot platforms with a common command interface. There exist several public and commercial software projects for interfacing with robots. Some examples are the following:

ARIA (ActivMedia Robotics Interface Application) is application programming interface developed under the object oriented model [18]. ARIA communicates with the robot via a client/server relationship through serial or TCP/IP connections, but it lacks buffering mechanisms. It is designed to interface commonnly with ActivMedia's mobile bases.

OROCOS (formerly Open Robot Control Software, now Open Real-time Control Software) is a free software project that focuses in real-time control [18]. It follows a component based design, where each component transmits their complete state in a single call. OROCOS is ideally suited for feedback control systems, but it is not an interface for higher level descriptions (such as joint and link configuration and geometry). Neither is OROCOS intended for motion playback under network limitations.

OAP (Open Automaton Project) focuses on hardware implementation. The goal of OAP is to provide inexpensive designs or building robots [8]. The Open Pino Project is a GNU project designed to control commercial Pino robots [8]. Thus, it lacks the potential to be a general platform to access other humanoids, mobile robots or human models.

Open HRP (Open Architecture Humanoid Robotics Platform) provides an abstraction layer between the hardware details of the robot and its controller systems [8]. The same controller

can be used on both real and simulated versions of the robot. RoboTalk adopts the same philosophy of hardware abstraction, but expands the class of robots to non-humanoids and provides different modes of network communication independently from the target robot.

Main Menu

Each main menu can be accessed by clicking on its name or using the ALT key and the first letter of the menu name. Once the menu appears on the screen, just click on the name of the function you want to use. Some functions have a letter in the name, which is underlined for quick keyboard access. Some functions can be accessed without the use of the pull down menus by certain CTRL key combinations, keyboard function keys, or the ESCAPE key.

When the user loads RoboTalk for Windows, the main editor screen comes up and the user is prompted to initialize the robot. If the user answers yes, the Mark IV controller performs a HARDHOME on the robot, and the RoboTalk for Windows software sets this position as the zero home position. If a HARDHOME has been performed before opening RoboTalk for Windows and the user answers no to the initialization prompt, the software uses the HARDHOME position as the zero home position. If the robot has not performed a HARDHOME, the user loads RoboTalk for Windows, and the user answers no to the initialization prompt; RoboTalk for Windows takes the position, the robot was in when the Mark IV controller was turned on, as home until a HARDHOME is performed. After the initialization procedure is complete, the Display box appears and the main editor screen becomes active.

The TITLE BAR will highlight, RoboTalk for Windows will put the program name "Untitled -RoboTalk" in the TITLE BAR, and the cursor will start blinking at the upper left hand corner of the editor screen. This signifies RoboTalk for Windows is ready to begin writing a new program. This is also how RoboTalk for Windows sets up the Editor screen, when the user chooses NEW from the FILE pull down menu or the NEW from the toolbar.

MENU OPTIONS

File Opens, Saves or Prints a program; Exits RoboTalk for Windows

Edit Undo, Cut, Copy, Paste, Find or Replace in program

View View Toolbar or Status Bar, change program Font

Robot Set modes of operation, displays and configuration

Help Help topics and version number

File Menu

- New
- Open
- Save
- Save As
- Print
- Print Preview
- Print Setup
- Exit

New

Clears the editor screen, sets the cursor in the upper left hand corner, sets the program name to "Untitled", and activates the editor screen.

Open

Displays the standard WINDOWS open box with the files under RoboTalk for Windows. The RoboTalk for Windows diskette or CD sent from the factory has example programs in a "Programs" subdirectory. This is where most RoboTalk for Windows programs will be saved. Only subdirectories and .rt files will show unless the user changes the bottom setup. Clicking on the program icon or typing in the program name will load the program to the editor screen. The open box can also be used to search for RoboTalk for Windows files in other directories.

Save

Quickly saves the program on the editor screen with the name in the TITLE BAR. **Be very careful with this menu item!** This save procedure provides no prompts and will save over the old file with the same name.

Save As

This menu item displays a Saves AS box, which allows the user to change the program name and the directory where the program is saved, before the program is saved.

Print

This item brings up the standard print box, allowing the user to choose the print options and parameters before printing a hard copy of the program on the editor screen.

Print Preview

This item allows viewing how the printed document will appear prior to actual printing.

Print Setup

The Print Setup Box is brought up so the user can adjust the printer's parameters.

Exit

This option will close down the RoboTalk for Windows software.

Edit Menu

- Undo
- Cut
- Copy
- Paste
- Find
- Find Next
- Replace

Undo

This allows the user to undo the last edit change.

Cut

This deletes the highlighted area and puts it on the clipboard.

Copy

This puts the current highlighted area onto the clipboard without deleting it from the program.

Paste

This places the information that is in the clipboard into the program at the current cursor position.

Find

This opens a Find box. Enter the text you want to search for and click "Find Next." The search can be case sensitive. It will search up or down.

Find Next

Searches for the next occurrence of the last find. This does not open a dialog box.

Replace

Opens a dialogue box that allows finding and replacing selected items with new entries. This function can also be case sensitive. There is also a global replace.

View Menu

- Tool bar
- Status bar
- Font

Toolbar

Displays Rhino Toolbar which includes the following buttons: New, Open, Save, Cut, Copy, Paste, Print, About, Help, Host, Play, Do, Run, Step, Pause, Stop, Display and Display Variables.

Status bar

Displays the Status bar which includes a brief message and the on-off status of HOST, CAP, NUM and SCROLL.

Font

This opens a dialog box which allows the user to change the type, style and size of the program text.

Robot Menu

- Connect to host
- Back to play mode
- Teach
- Do
- Run
- Pause
- Stop Running
- Display Dialog
- Display Variables
- Point Data
- Configure
- Highlight Tracing
- Load data file

Connect to Host

This allows the RoboTalk for Windows computer software to take control of the Mark IV controller and robot. The robot gripper will close and open, then RoboTalk for Windows will ask the user if the robot should be initialized.

Back to Play Mode

This gives control of the robot to the Mark IV controller. All RoboTalk for Windows functions work except the ones which require communications with the Mark IV. In other words, the user can edit programs and data points, but not send commands or receive replies from the Mark IV controller. The robot can only be controlled through the Mark IV hand unit or Teach Pendant. The program can be run and it will ignore commands for communications to and from the Mark IV controller.

Teach

This mode allows the user to operate certain keys on the Mark IV Teach Pendant Hand to add taught points to the current program. The available keys are - all **MOTOR** keys, the **UP** and **DOWN ARROWS**, the **OPEN**, **CLOSE**, **ENTER**, **ESCAPE** and **STOP** keys.

A dialog box appears for naming the new point. Name the point and click done. After positioning the robot with the Hand, press ENTER on the Mark IV hand unit to add the new point to the program's point data file. If the user saves more than one robot position, the additional points will have the same point name with a sequential number added to the end. RoboTalk for Windows will also add a MOVEP command with the new point name, to the program, at the cursor position.

The OPEN and CLOSE keys are used to open and close the gripper. If the gripper changes state, RoboTalk for Windows will add an OPEN or CLOSE command at the cursor. If the gripper does not change state RoboTalk for Windows will add a MOVEP command, at the cursor. Many points can be added in this manner. All points whose names begin with an X will be saved in Cartesian coordinates.

To end the TEACH mode, press the ESCAPE key on the Mark IV Hand unit. This will bring you back to the regular edit mode. The new points will be stored in the point data file, where they can be further modified or added to other locations in the program, using the MOVEP commands.

Do

The Do mode opens a dialog box, which allows the user to type in and directly execute one command at a time. Most RoboTalk for Windows commands can be executed in the Do mode. Commands such as GOSUB, which require more than one line of code, cannot be used in the direct mode.

The Do mode places RoboTalk for Windows in the direct execution mode where individual commands can be executed without being used in the context of an application program.

Direct execution is useful when debugging a new work cell system, where the effects of the various input and output setups can be evaluated. In addition the robot can be moved with the MOVE and MOVE TO commands, allowing the user to study the effects of various paths between endpoints. After the Do mode is finished executing the command, the Do dialog box will return to the screen ready for the next command.

Run

The run mode opens a dialog box which allows the user to choose how many times the program should repeat. Choose OK to run the program. **Caution!** The robot will return to the home position before the program starts.

Pause

Temporarily halts the program after finishing the current line of code that is executing. To continue, choose run.

Stop Running

Halts the program, when the program is finished executing the current line of code. The editor screen will be active.

Display

When RoboTalk for Windows is first opened, communications is established, the robot gripper is closed and opened, and an initialization dialog box appears on the screen. If the user answers yes, RoboTalk for Windows performs a HARDHOME routine. When the

HARDHOME routine is finished, the Display box appears on the screen. If the user answers no in the initialization dialog box; no HARDHOME routine occurs and the Display box appears on the screen.

The user can close the Display box. Choosing the Display option will make the Display box reappear. If another dialog box is activated or if the editor screen is activated, choosing the Display option will reactivate the Display box. The Display box has a screen at the top which displays messages from the program.

When the trace function is on, the Display box shows the current position of the robot and the status of the input, output, and auxiliary ports. The bottom of the Display box alerts the user when RoboTalk for Windows is in Teach mode and the name that will be used for the taught point.

Display Variables

This opens a dialog box which displays the values of the 26, A-Z, integer variables available in RoboTalk for Windows.

Point Data

This option opens up the Robot Point Data box. This box displays all the taught points for the program. These points are entered in the Teach mode using the Mark IV Teach pendant hand. The box is set up for two types of points Joint and Cartesian. The Joint positions use motor encoder counts for robot position values. The Cartesian positions use XYZ coordinates for robot position values.

The **Add** button inserts a new name in the taught point list and gives this point the current position of the robot.

The **Teach** button allows the user to teach this new point or modify an existing highlighted point using the Mark IV hand unit. The Teach button acts the same as the Teach mode option but does not insert a MOVEP command into the program.

The **Delete** button will remove a highlighted taught point from the data file.

The **Modify** button opens a dialog box which allows the user to enter new values for encoder counts in Joint positions or new values for coordinates in Cartesian positions for the highlighted point.

The **Move** button will move the robot to the position of the current highlighted point in the list.

The **Reload** button will reload the original data file, if a save has not been performed. This will be the data file with the same name as the program file.

The **Save** button saves the changes made in the Point Data box.

The **Print** button will print a hard copy list of all the points in the Point Data box including every encoder or coordinate value for each point.

The **Exit** button will close the Robot Point Data box and return to the editor screen.

Configure

The configure option allows the user to set up RoboTalk for Windows in various operating modes with various parameter settings. These settings will be stored in the computer, so RoboTalk for Windows will default to these settings when it is started again. When the user chooses the Configure option, a Robot Configuration box appears on the screen. This box allows the user to set the modes and parameters as follows:

Serial Port: NONE, COM1, COM2, COM3, COM4
Sets the computer to communicate with the Mark IV controller through port 1, 2, 3 or 4. Also, separate IO MODULE unit can be set to communicate through one of these ports. Both units cannot be set to the same port.

Mode: SCARA or XR
This allows RoboTalk for Windows to use the proper routines and parameters when controlling robot, which is connected to the Mark IV. One such as routine is the

HARDHOME routine, which is used to initialize the robot. Make sure this parameter is set for the type of robot that is being used.

Controller Type: Mark III or Mark IV.

This is used to set the type of controller that will be used with RoboTalk for Windows. The Mark IV controller has an encoder resolution which is four times greater than the Mark III. The Mark IV uses some commands in a slightly different format than the Mark III and RoboTalk for Windows has an expanded set of commands which will only be recognized by the Mark IV. Programs taught using the Mark IV will not necessarily work with the Mark III and vice versa. The MOVE commands will not place the robot in the same positions and possibly cause crashes. Some commands will cause syntax errors or unknown command messages. Be sure this setting corresponds to the controller that is being used.

I/O Module: Offline or Online

Connects or disconnects communications with the serial port set in the box above. This relates to a separate Rhino I/O Module unit. The I/O Module unit offers control of various digital and analog input and output ports, which could be very helpful in work cell design, set up and experimentation. RoboTalk for Windows will have a set of commands to control this unit through a computer. This is not related to the input and output ports on the front of the Mark III or Mark IV and is only used with the Rhino I/O Module unit. The communications port cannot be the same port the Mark IV is using.

Robot: Online or Offline

Connects or disconnects communications to the robot controller with settings from the boxes above. The same actions can be accomplished using the "Connect to Host" and "Back to Play Mode" functions in the Robot Menu or the Host and Play buttons on the Tool Bar. When RoboTalk for Windows connects communications with the Mark IV controller, it opens and closes the gripper and asks if the robot should be initialized.

Gripper Mode: Motor, Output or Aux

This setting allows the OPEN and CLOSE commands to be directed away from the robot

gripper A motor to OUTPUT 1 PORT or the AUX 1 PORT. If a non standard gripper, such as a vacuum end effector is being used, the OPEN and CLOSE commands will activate the user's choice of either OUTPUT 1 PORT or AUX 1 PORT. A CLOSE command will turn on the port and an OPEN command will turn off the port. Care should be exercised in using the AUX 1 and OUTPUT 1 modes since other RoboTalk for Windows commands still control the AUX 1 OUTPUT 1 ports.

Speed Compensation: 1 - 50.

This parameter is used only in the Mark III mode and has no effect when RoboTalk for Windows is in the Mark IV mode. The Mark III works at a slower frequency than most computers. Use the lowest setting which will give consistent communications when using the Mark III.

XYZ Parameters: Home and Offset Parameters and Units.
This is the last selection under the Configure option. This button opens up a separate dialog box. The dialog box has various parameters for Cartesian coordinates. The parameters are divided into XR and SCARA sections. The parameters set the Cartesian coordinates of the robot home position and the offset position of the user's coordinate system compared to the robot's coordinate system. Usually the user and robot coordinate systems are exactly the same. See the chapter on Cartesian coordinates for further details. This dialog box also allows the Cartesian coordinates to be set for millimeters or inches. The default setting from the factory will be set to millimeters. BE CAREFUL NOT TO RUN A PROGRAM, THAT USES MILLIMETERS IN THE MOVE COMMANDS, WITH ROBOTALK FOR WINDOWS SET TO INCHES, OR VICE VERSA.

Highlight Tracing

This function will reverse highlight the current executing line during the Run mode.

Load data file

Each RoboTalk for Windows program file has a related data file with the same name and a " .DAT" extension. When a program file is opened, the information in the corresponding data file is made available through the Point Data dialog box. The Load data file function allows the program use of a data file from another program. This would be helpful if the user was starting a new program which will use many robot positions of a previous program. **Caution: Loading a different data file and saving will overwrite the original data file!**

If one data position from a previous program is needed, use the following procedure:

1. Open the program that needs a point from another program.
2. Click on the Robot menu and choose the Load Data File Function. Click on YES.
3. Choose the .DAT file which has the point that you want to copy.
4. Click on the Robot Menu and choose the Point Data Function.
5. Highlight the name of the point to be copied and click on the MOVE button. Click on the EXIT button. **DO NOT SAVE!**
6. Click on the Robot Menu and choose Load Data File again. Click on YES.
7. Choose the original .dat file that goes with the program.
8. Click on the Robot menu and choose Point Data.
9. Click on the ADD button in the correct section - Joint or Cartesian.
10. Type in a name for the point and click on the OK button.
11. Click on the SAVE TO THE FILE button.

PROGRAM CONTROL COMMANDS

BEEP Produce a short tone

CLS Clear screen

END Exit program

ERROR_CHECK List Mark IV error stack contents

FOR <index> = <start> TO <end> NEXT Program loop

GET_ANSWER <string> Interrogate Mark IV

GOSUB <label> Call subroutine

GOTO <label> Unconditional branch/jump

IF <exp1> <condition> <exp2> THEN <statement> Conditional execution

IFMICRO <motor letter> THEN <statement> Conditional execution

IFSIG <signal #> [, <signal #>...] THEN <statement> Conditional execution

INPUT ["string",] <var> Input from computer keyboard

LOCAL <var> [, <var>...] Define local subroutine variable

PAUSE <var> Delay for <var> * 0.1 second

REM <comment> Comment

RETURN Return from subroutine

SEND <string> Send <string> to Mark IV

SETI <var> = <integer> Set integer

TROFF Trace function off

TRON Trace function on

TYPE <string> [;] or <numeric exp> [;] Output to screen

WAITFOR <input #> [, <input #>...] Wait for signal condition

ROBOT and I/O CONTROL COMMANDS

ARC <point1>, <point2>[, <loops>][,<velocity>][, <max move>] Mark IV Point defined circular move

ARCX <X1>, <Y1>, <Z1>, <X2>, <Y2>, <Z2> [, <loops>][, <velocity>][, <max move>] Mark IV Cartesian circular move

AUX <port num> Aux port control, Mark III

AUX <port num> = <var> Aux port control, Mark IV

CLEARPATH Mark IV Clear continuous path

CLOSE [<var>] Close gripper

GRIP Close gripper, record in Z

HARDHOME Find all microswitches

HOME Move to home position

MOVE <B>, <C>, <D>, <E>, <F> Incremental joint move

MOVE TO <B>, <C>, <D>, <E>, <F> Absolute joint move

MOVEGH <G>, <H> Incremental move

MOVEGH TO <G>, <H> Absolute move

MOVEP <teach point joint location> Move to joint location

MOVES <point> [, <velocity>][, <max move>] Mark IV Point defined straight line move

MOVEXS <X>, <Y>, <Z> [, <velocity>][, <max move>] Mark IV Cartesian straight line move

MOVEX <X>, <Y>, <Z>, <A>, <T> Incremental Cartesian move

MOVEX TO <X>, <Y>, <Z>, <A>, <T> Absolute Cartesian move

MOVEXP <teach point Cartesian location> Move to a Cartesian point location

OFFLINE <motor letter> [, <motor letter>...] Ignore a motor

OFFSET <motor letter> = <var> Joint (encoder) position offset

OFFSETX <Cartesian axis> = <var> Cartesian position offset

ONLINE <motor letter> [, <motor letter> ...] Activate an offline motor

OPEN [<var>] Open gripper

OUTSIG <signal #> [, <signal #>...] Output signal line control

PATH [<velocity>][, <max move>] Mark IV Continuous path move

POINT <location> Set <joint location point> to current position

POINT <location> Set <Cartesian point> to current position

POINT <location> = <B>, <C>, <D>, <E>, <F> Set <joint location point>

POINT <location> = <X>, <Y>, <Z>, <A>, <T> Set <Cartesian location point>

SEARCH <motor letter>, <dir/step>, <port num> Conditional move

SETPATH <point> Mark IV Set continuous point

SETPATHX <X>, <Y>, <Z> Mark IV Set continuous Cartesian point

VEL <motor letter> = <var> Set Mark IV motor velocity

7. REFERENCES

7.1 Literature References

1. "Robotic Manipulation and the Product of Exponentials Formula," Mathematical Theory of Networks and Systems, R. W. Brockett, (P. A. Fuhrman, Ed.) Springer 2005 32(4):430-35.

2. Development of Metal-Mold Polishing Robot System with Contact Pressure Control Using CAD/CAM Data Y. Mizugakia, M. Sakamotoa, K. Kamijob, N. Taniguchia Kyushu Institute of Technology/ vol.23, no.6. July 2001. pp.48-81

3. Development of high power laser pipe welding process. Ono M; Shimbo Y; Ohmura M; SekineY; Iwasaki K; Takahashi M. Quarterly Journal of the Japan Welding Society, vol.19, no.2.May 2001. pp.233-240.

4. CAD/CAM-based position/force controller for a mold polishing robot by Fusaomi Nagataa,Tetsuo Haseb,Zenku Hagab, Masaaki Omotob, Keigo Watanabec by Kyushu Kyoritu University/ Vol. 5 No. 2 April 2002.

5. Hirukawa, T.Matsui and K.Takase, Automatic determination of possible velocity and applicable force of frictionless objects in contact from a geometric model, IEEE Trans. Robotics and Automation, pp.309-322, 1994.

6. H.Inoue, S.Tachi, Y.Nakamura, K.Hirai, N.Ohyu, S.Hirai,K.Tanie, K.Yokoi and H.Hirukawa, Overview of Humanoid Robotics Project of METI, Proc. of the 32nd ISR, April,2001.

7. J. Knani, Dynamic modelling of flexible robotic mechanisms and adaptive robust control of trajectory computer simulation – Part I,Appl. Math. Model. 26 (2002) 1113–1124.

8. D.N.C. Tse, P. Viswanath, L. Zheng, Open Architecture robot controller with diversity and multiplexing trade off in multiple-access channels, IEEE Transactions on Information Theory 50 (9) (2004).

9. Development and application of an intelligent welding robot system for shipbuilding. Donghun Lee n, Namkug Ku, Tae-Wan Kim, Jongwon Kim, Kyu-Yeul Lee, Youg-Shuk Son, August 2010

7.2 Books References

10. Schilling J Robert, Fundamentals of Robotics "Analysis and Control" by Prentice Hall 2005.

11. Craig J John, Introduction to Robotics "Mechanics and control" Third edition by Pearson Education 2009.

7.3 Web References

12. Introduction to Rhino Robot [Online]. Available at http://www.rhinorobotics.com/xr3_flyer.html (Accessed: 16 November 2011)

13. Various Configurations of Robot [Online]. Available at http://www.google.co.in/imgres?q=configuration+of+robot&um=1&hl=en&biw=1366&bih=659&tbm (Accessed: 14 November 2011)

14. Current programming approaches [Online].Available at http://dl.acm.org/citation. cfm?id=2051514. (Accessed: 06 Febuary 2012).

15. Motion-Control Performance Requirements [Online].Available at http://alexandria. tue.nl/extra2/200411174.pdf (Accessed: 06 Febuary 2012).

16. Controller of rhino robot [Online] http://www.rhinorobotics.com/markiii.html (Accessed: 23 November 2011).

17. Rhino Robotics Xr-3 Motor Encoder Optics [Online]. Available at http://www.rhinorobotics.com/ xrencoder.html. (Accessed: 23 November 2011).

18. Additional Technical Information of rhino robot [Online]. Available at: http://www.postech.ac.kr/ class/ee565/lab/material20.pdf. (Accessed: 21 December 2011).

19. Introduction to Peripheral Interface Controller [Online]. Available at http://www.circuitstoday.com/ peripheral-interface-controller-pic. (Accessed: 10 January 2012).

20. Why PIC Chosen for Servomotor Control [Online]. Available at http://www.mikroe.com/eng/chapters/view/15/chapter-2-programming-microcontrollers. (Accessed: 18 January 2012).

21. PIC Specifications [Online]. Available at http://www.microchip.com/stellent/idcplg? IdcService=SS_GET_PAGE&nodeId=2551. (Accessed: 18 January 2012).

22. Introduction of PICFlash [Online]. Available at http://www.microchip.com/stellent /idcplg/IdcService=SS_GET_PAGE&nodeId=2617&param=en533117. (Accessed: 22 January 2012).

23. Programming a Microcontroller [Online].Available at http://www.mikroe.com/eng/chapters/view/11/appendix-a-programming-a-microcontroller. (Accessed: 26 Febuary 2012).

24. Polulu Orangutan Controller [Online]. Available at http://www.pololu.com/catalog/category/37. (Accessed: 04 May 2011).

25. Arbotrix Controller [Online]. Available at http://www.trossenrobotics.com/p/phidgets-motor-control-1motor.aspx (Accessed: 04 May 2011).

26. Arduino Controller [Online]. Available at http://arduino.cc/en/Main/ArduinoBoardADK (Accessed: 04 May 2011).N

27. Rhino robotics pin outs [Online]. Available at http://www.rhinorobotics.com/pinout.html (Accessed: 23 November 2011).

28. Introduction to cylinder liner. Available at http://chianzzdh.win.mofcom.gov.cn/www/8%5Cchianzzdh%5Cimg%5C2008102416442.jpg (Accessed: 18 November 2011)

29. Cylinder liner welding. Available at http://www.sulzer.com/en/media/Media/Images/ProductsAndServices/CoatingServices/Thermal_Spray/Industriestransportationb.jpg?mw=690 (Accessed: 18 November 2011)