Sudan University of Science & Technology

College of Graduate Studies

DESIGNING AND BUILDING A MODEL FOR VISUAL SERVOING ROBOTIC ARM

تصميم وبناء نموذج ذراع روبوت تآزري بصري

A Thesis Submitted in Partial Fulfillment to the Requirements for the Degree of M.Sc. in Mechatronics Engineering

Prepared by: Muhanad Muatasim Osman Elnour

Supervised by: Dr. Musaab A.H. Zaroug

November 2015

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

DEDICATION

Dedicated to
My FAMILY

ACKNOWLEDGEMENT

I would like to thank all those who supported me, my mother, my father, my brothers, my sisters and my friends. Special thanks are due to my Supervisor: Dr. Musaab A.H. Zaroug, for his supporting me. I greatly express my thanks to all persons whom supported me in preparing this research.

ABSTRACT

Visual Servoing has been a focus of intensive research form the last three decades due to its various robotic applications, mainly to allow performing autonomous tasks with the help of image sensors.

This thesis presenting a visual servoing control system for a 3DOF robotic arm which is visual approach of servo using a vision sensor as a feedback, this approach introduce a better flexibility more than usual used passive sensors allow the robot arm to track the object without depending on a predefined location

In this thesis a 3DOF robotic arm built and servo motors and Arduino Board used to control the moving of the robotic arm. An analytic calculations made for the robot to define the DH parameters and to get the forward and inverse kinematics equations for the robot.

An Object detection algorithm developed to extract an object center from an image. A calibration method developed to find the scales between the image pixels and the dimensions on the robot real world.

And to perform the visual servoing an algorithm developed which initialize the system, calibrate it, finding the object in the photo, convert it to a position and send it to the robot to track the object. The system successfully built and tested and could track the object

الملخص

خلال الثلاثة عقود الماضية كان مجال المؤازرة البصرية (Visual Servoing) مجال بحث مكثف من قبل الباحثين وذلك لتطبيقاته المتعددة في مجال الروبوتات وبشكل اساسي لتمكينه من تنفيذ مهام بشكل مستقل (دون الحوجة للتدخل البشري) وذلك باستخدام الحساسات البصرية .

تقدم هذه الأطروحة نظام مؤازرة بصري للتحكم في ذراع روبوتية ذات ثلاثة درجات حرية عبارة عن نظام مؤزرة باستخدام حساس بصري كتغذية عكسية , تضيف هذه الطريقة مرونة أعلى من استخدام الحساسات العادية مما يسمح للروبوت بتتبع جسم بشكل مستقل من دون تحديد موقعه مسبقا .

في هذه الأطروحة تم بناء ذراع روبوتية بثلاث درجات حرية , تم استخدام محركات تآزرية بالاضافة الى لوحة اردوينو للتحكم في حركة الروبوت , كما تم عمل الحسابات اللازمة لتحديد معاملات (DH) للذراع الروبوتية واستخراج معادلات الحركة للروبوت .

تم تطوير خوارزمية تقوم باستخراج مركز الجسم المراد تتبعه من الصورة كما تم تطوير طريقة معايرة تسمح باجاد معاملات العلاقة بين عناصر الصورة والابعاد الحقيقية لبيئة الروبوت .

ولتنفيذ المؤازرة البصرية تم تطوير خوارزمية تقوم بتهيئة النظام ومعايرته ومن ثم ايجاد الجسم من الصورة واستخراج مركزه من بيئة الروبوت ومن ثم ارساله الى الروبوت لتتبعه , تم تصميم وتنفيذ هذا النظام واختباره لتتبع جسم بشكل ناجح .

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION AND LITERATURE REVIEW

CHAPTER ONE
INTRODUCTION AND LITERATURE REVIEW

1.1 Introduction

Visual servoing, also known as Vision-Based Robot Control and abbreviated VS, is a technique which uses feedback information extracted from a vision sensor to control the motion of a robot. One of the earliest papers that talked about visual servoing was from the SRI International Labs in California, USA (Founded as Stanford Research Institute SRI became independent in 1970) [1]. A first tutorial on Visual Servoing was published in 1996 by S. A. Hutchinson, G. D. Hager, and P. I. Corke [2], more recent tutorials were published in 2006 and 2007 by F. Chaumette and S. Hutchinson[3][4]. A collection of state-of-the-art research is a published work by Chesi and Hashimoto (2010). Focusing on advanced numerical methods, it is divided in three main fields: Vision, Estimation and path planning and Control [5].
Robot manipulator is one of the motivating disciplines in industrial and educational applications, and an essential branch to control sciences because of its intelligent aspects, nonlinear characteristics, and its real time implementation. It was developed to enhance human's work such as in the manufacturing or manipulation of heavy materials, and unpredictable environments. Whatever the kind of task robot manipulator may be provided with, robot performance measures the high quality and large quantity of work that it can do in the desired time and place. Robot manipulator has immeasurable tasks, so it is designed to be flexible in general motions to move from one position to another with smooth movement to avoid sharp jolt in the robot arm, These jolts may damage the arm.
Normally there are three main subsystems in robot manipulators: mechanical system, electrical system, and control system. Mechanical system comprises of all movable parts. It consists of a group of links (rigid bodies) connected together by joints which allow the motion for the desired link. The mechanical system is used to move the end effector (the top link) to xyz position with respect to the base. This movement depends on the electrical system (e.g. motors, power amplifiers, and other electronic circuits) and it is done by some rotations and translations to the other links. Due to the variety of tasks and duties in robot manipulators, its construction is divided into two main classes: serial manipulator and parallel manipulators [6].
Serial manipulators consist of some links connected in series, which form an open loop chain. At the end of the chain, the end effector is connected to the base by single kinematic chain. On the other side, the parallel manipulators form a closed loop chain finished by the end effector and is connected to the base by two, or more kinematic chains (e.g. arm, or legs). The only drawback of the parallel manipulator over the serial manipulator is that the parallel robots manipulators suffer from limited workspace as compared with serial robot manipulators.

1.2 Related Work

This section gives an overview of recent research results in the domain of visual servoing. The term visual servoing has been introduced in 1979 by Hill and Park [1]. Since then the rapid technological development of cameras, computers and robots has opened up possibilities for improvements as well as new approaches and ideas. A recent comprehensive overview of all techniques that include visual systems in robotics is given by Kragic and Vincze [7]. Besides the general overview they also provide introductions to specific fields, such as visual servoing. Furthermore, they cluster the existing branches into already working techniques and still open challenges.
Another recent survey about the domain of visual servoing is presented by Kragic and Christensen [8]. In this survey they summarize and classify about 100 different approaches according to the number of controlled degrees of freedom, the type of control model and the camera configuration. Hutchinson, Hager and Corke [2] as well as F. Chaumette and S. Hutchinson [3,4] also provide tutorials as an introduction to this field. This thesis is primarily based on these tutorials, but uses also ideas and techniques of other approaches.
And newer survey by Ricardo Manuel Candeias da Silva Santos [9] for evaluating Visual Servoing Techniques for Robotic Applications with the goal to elucidate in which situations each one of them is a good option to solve a specific task. The methods evaluated are: position based visual servoing, image based visual servoing, 2.5D visual servoing, decoupled visual servoing and shortest path visual servoing.
In recent year visual servoing was a field of heavy researches form implementation in various fields like “A Visual Servoing System for Interactive Human-Robot Object Transfer” [10], using an artificial intelligent “ANN Based Robotic Arm Visual Servoing Nonlinear System”[11] , to implement an uncalibrated visual servoing “Visual Task Specification User Interface for Uncalibrated Visual Servoing”[12].

1.3 Scope

The focus of this research is to building a robotic motion control using image processing. The research will focus on building and simulating of the tracking algorithms using MathWorks® MATLAB® tools [13,14,15].

1.4 Objectives

The objectives of the research can be achieved through the following:

1. Building computer simulation model of a robotic arm
2. Designing a visual servo system for the robotic arm
3. Implement a visual servo system for the robotic arm

1.5 Problem Statement

The control of robotic arm features (e.g. motion trajectory and arm Position) nowadays are built on either preprogramed actions or passive sensors (proximity, limit switches, … etc.) or both. This will introduce large flexibility in robotic application.
Using image processing instead of passive sensor (known as Visual Servoing) gives better flexibility for the robotic arms applications.

1.6 Methodology

The research work will focus on modelling and simulating the robotic arm system with one of the existing visual servoing image processing algorithm

1.7 Thesis Structure

The research outlines will be as follow:
Chapter One: Introduction and literature review
Chapter Two: Theoretical background
Chapter Three: System Component
Chapter Four: Implementation
Chapter Five: Results
Chapter Six: conclusions and recommendations

CHAPTER TWO

THEORETICAL BACKGROUND

CHAPTER TWO
THEORATICAL BACKGROUND

2.1 Introduction

2.1.1 What is Robot

According to the Encyclopaedia Britannica, a robot is "any automatically operated machine that replaces human effort, though it may not resemble human beings in appearance or perform functions in a humanlike manner". Merriam-Webster describes a robot as a "machine that looks like a human being and performs various complex acts (as walking or talking) of a human being", or a "device that automatically performs complicated often repetitive tasks", or a "mechanism guided by automatic controls".

A robot is a mechanical or virtual artificial agent, usually an electro-mechanical machine that is guided by a computer program or electronic circuitry. Robots can be guided, autonomous or semi-autonomous and range from humanoids to industrial robots, collectively programmed swarm robots, and even microscopic Nano robots. By mimicking a lifelike appearance or automating movements, a robot may convey a sense of intelligence or thought of its own.

The word 'robot' was first used to denote fictional humanoid in a 1921 play R.U.R. (Rossum’s Universal Robots) by the Czech writer, Karel Čapek. Electronics evolved into the driving force of development with the advent of the first electronic autonomous robots created by William Grey Walter in Bristol, England in 1948. The first digital and programmable robot was invented by George Devol in 1954 and was named the Unimate. It was sold to General Motors in 1961 where it was used to lift pieces of hot metal from die casting machines at the Inland Fisher Guide Plant in the West Trenton section of Ewing Township, New Jersey.

Robots have replaced humans in the assistance of performing those repetitive and dangerous tasks which humans prefer not to do, or are unable to do due to size limitations, or even those such as in outer space or at the bottom of the sea where humans could not survive the extreme environments.

2.1.1.1 Robots uses now a days

Robots can be classified by their specificity of purpose. A robot might be designed to perform one particular task, or a range of tasks. Of course, all robots by their nature can be re-programmed to behave differently, but some are limited by their physical form. For example, a factory robot arm can perform jobs such as cutting, welding, gluing, or acting as a fairground ride, while a pick-and-place robot can only populate printed circuit boards.

Robot now a days play main rule in many fields such as:

<u>Car production:</u>

Over the last three decades, automobile factories have become dominated by robots. A typical factory contains hundreds of industrial robots working on fully automated production lines, with one robot for every ten human workers. On an automated production line, a vehicle chassis on a conveyor is welded, glued, painted and finally assembled at a sequence of robot stations.

Packaging:

Industrial robots are also used extensively for palletizing and packaging of manufactured goods, for example for rapidly taking drink cartons from the end of a conveyor belt and placing them into boxes, or for loading and unloading machining centers.

Electronics:

Mass-produced printed circuit boards (PCBs) are almost exclusively manufactured by pick-and-place robots, typically with SCARA(Selective Compliance Assembly Robot Arm) manipulators, which remove tiny electronic components from strips or trays, and place them on to PCBs with great accuracy. Such robots can place hundreds of thousands of components per hour, far out-performing a human in speed, accuracy, and reliability.

Automated guided vehicles (AGVs):

Mobile robots, following markers or wires in the floor, or using vision or lasers, are used to transport goods around large facilities, such as warehouses, container ports, or hospitals

Space probes:

Almost every unmanned space probe ever launched was a robot. Some were launched in the 1960s with very limited abilities, but their ability to fly and land (in the case of Luna 9) is an indication of their status as a robot. This includes the Voyager probes and the Galileo probes, and others.

Domestic robots:

Domestic robots are simple robots dedicated to a single task work in home use. They are used in simple but unwanted jobs, such as vacuum cleaning, floor washing, and lawn mowing. An example of a domestic robot is a Roomba.

Military robots:

Military robots include the SWORDS (Special Weapons Observation Reconnaissance Detection System) robot which is currently used in ground-based combat. It can use a variety of weapons and there is some discussion of giving it some degree of autonomy in battleground situations.

Mining robots:

Mining robots are designed to solve a number of problems currently facing the mining industry, including skills shortages, improving productivity from declining ore grades, and achieving environmental targets. Due to the hazardous nature of mining, in particular underground mining, the prevalence of autonomous, semi-autonomous, and tele-operated robots has greatly increased in recent times

Healthcare:

Robots in healthcare have two main functions. Those which assist an individual, such as a sufferer of a disease like Multiple Sclerosis, and those which aid in the overall systems such as pharmacies and hospitals.

2.1.1.2 Robotic arm (manipulator)

A robotic arm is a type of mechanical arm, usually programmable, with similar functions to a human arm; the arm may be the sum total of the mechanism or may be part of a more complex robot. The links of such a manipulator are connected by joints allowing either rotational motion (such as in an articulated robot) or translational (linear) displacement. The links of the manipulator can be considered to form a kinematic chain. The terminus of the kinematic chain of the manipulator is called the end effector and it is analogous to the human hand.

Robotic hand:

The end effector, or robotic hand, can be designed to perform any desired task such as welding, gripping, spinning etc., depending on the application. For example robot arms in automotive assembly lines perform a variety of tasks such as welding and parts rotation and placement during assembly

Robotic Arm Types:

- Cartesian robot / Gantry robot: Used for pick and place work, application of sealant, assembly operations, handling machine tools and arc welding. It's a robot whose arm has three prismatic joints, whose axes are coincident with a Cartesian coordinator.
- Cylindrical robot: Used for assembly operations, handling at machine tools, spot welding, and handling at diecasting machines. It's a robot whose axes form a cylindrical coordinate system.
- Spherical robot / Polar robot (such as the Unimate): Used for handling at machine tools, spot welding, diecasting, fettling machines, gas welding and arc welding. It's a robot whose axes form a polar coordinate system.
- SCARA robot: Used for pick and place work, application of sealant, assembly operations and handling machine tools. This robot features two parallel rotary joints to provide compliance in a plane.
- Articulated robot: Used for assembly operations, diecasting, fettling machines, gas welding, arc welding and spray painting. It's a robot whose arm has at least three rotary joints.
- Parallel robot: One use is a mobile platform handling cockpit flight simulators. It's a robot whose arms have concurrent prismatic or rotary joints.
- Anthropomorphic robot: Similar to the robotic hand Luke Skywalker receives at the end of The Empire Strikes Back. It is shaped in a way that resembles a human hand, i.e. with independent fingers and thumbs.

2.1.2 The Senses

A sense is a physiological capacity of organisms that provides data for perception. The senses and their operation, classification, and theory are overlapping topics studied by a variety of fields, most notably neuroscience, cognitive psychology (or cognitive science), and philosophy of perception. The nervous system has a specific sensory system or organ, dedicated to each sense.

Humans have a multitude of senses. Sight, hearing, taste, smell, and touch, are the five traditionally recognized, The ability to detect other stimuli beyond those governed by these most broadly recognized senses also exists, and these sensory modalities include temperature , kinesthetic sense, pain, balance, vibration, and various internal stimuli.

Other animals also have receptors to sense the world around them, with degrees of capability varying greatly between species. Humans have a comparatively weak sense of smell relative to many other mammals while some animals may lack one or more of the traditional five senses. Some animals may also intake and interpret sensory stimuli in very different ways. Some species of animals are able to sense the world in a way that humans cannot, with some species able to sense electrical and magnetic fields, and detect water pressure and currents.

2.1.2.1 Visual perception

Visual perception is the ability to interpret the surrounding environment by processing information that is contained in visible light. The resulting perception is also known as eyesight, sight, or vision, and are the focus of much research in psychology, cognitive science, neuroscience, and molecular biology, collectively referred to as vision science

2.1.3 Visual Servoing

Visual servoing is a robot control method in which visual feedback or image from camera sensors is introduced into the robot inside a control loop to accomplish tasks in unstructured environments and to enhance the robot control performance [2]. Visual servoing can be categorized into image-based visual servoing (IBVS), position-based visual servoing (PBVS), and hybrid approach [16]. In the visual servoing systems, there are two basic camera configurations. One is to install a camera at the tip or end-effector of the manipulator (eye-in-hand); the other is to set the camera and manipulator separately (eye-to-hand) [17].

Compared with the other visual servoing methods, IBVS has three main advantages and has gained research interest among the robotics community since it is insensitive and robust to camera model error, calibration error, and measurement noise [18]. Hence IBVS is widely used for automated robotic manufacturing systems [19].

But unfortunately, it is known that the interaction matrices of IBVS actually depend on the depths of target features [20]. In most of cases, target feature depth is treated as constant or just roughly estimated such that the visual servoing system remains stable and convergent because of the robustness of IBVS to the error of camera model and system calibration. Nevertheless, in some cases the incorrect depth estimation of target features may cause convergence and stability problems [21]. Therefore, the exact determination of target feature depths is still a crucial task in the design of IBVS systems. Moreover, in large-scale manufacturing systems, the dynamic forces of the workpiece play an important role in robot dynamic behaviors, since the mass of large size workpiece is not negligible.

In the determination of image features in visual servoing, geometric features such as points, segments or straight lines [2] are usually chosen as the target features, and the corresponding image features in the image plane are utilized as image features and the inputs of visual servoing controllers. However, such target features can only be applied in certain limited target objects [22]. In addition, the geometric features might be occluded from the FOV of the camera. In this case, the number of the image features does not match with that of the desired ones, which may lead to the failure of visual servoing. In order to track the target objects which do not have enough detectable geometric features, and to enhance the robustness of visual servoing systems, several novel target features are adopted for visual servoing. For example, laser points and the polar signature of target object contour are used as target features in the design of IBVS systems.
The image moments are normally used for pattern-recognition in computer vision [23, 24]. Recently they have been adopted as image features for visual servoing control scheme design due to their easy computation from binary or segmented image or from a set of extracted points of interest, disregarding the target object shape complexity and their generic representation of any target object, with a simple or complex shape [25]. In addition, low-order moments have an intuitive meaning, since they are directly related to the area, the centroid, and the orientation of the object in the image plane [26]. Following previous work, it is known that using the image moments as image features in visual servoing can render the corresponding interaction matrix with maximal decoupled structure, and the inherent problem-singularity of the interaction matrix is avoided and the control performance of IBVS system is thus improved.

2.1.4 Matlab Platform

MATLAB is a platform for scientific calculation and high-level programming which uses an interactive environment that allows you to conduct complex calculation tasks more efficiently than with traditional languages, such as C, C++ and FORTRAN. It is the one of the most popular platforms currently used in the sciences and engineering.

MATLAB is an interactive high-level technical computing environment for algorithm development, data visualization, data analysis and numerical analysis. MATLAB is suitable for solving problems involving technical calculations using optimized algorithms that are incorporated into easy to use commands.

It is possible to use MATLAB for a wide range of applications, including calculus, algebra, statistics, econometrics, quality control, time series, signal and image processing, communications, control system design, testing and measuring systems, financial modeling, computational biology, etc. The complementary toolsets, called toolboxes (collections of MATLAB functions for special purposes, which are available separately), extend the MATLAB environment, allowing you to solve special problems in different areas of application.

In addition, MATLAB contains a number of functions which allow you to document and share your work. It is possible to integrate MATLAB code with other languages and applications, and to distribute algorithms and applications that are developed using MATLAB.

The following are the most important features of MATLAB:

- It is a high-level language for technical calculation
- It offers a development environment for managing code, files and data
- It features interactive tools for exploration, design and iterative solving
- It supports mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, and numerical integration
- It can produce high quality two-dimensional and three-dimensional graphics to aid data visualization
- It includes tools to create custom graphical user interfaces
- It can be integrated with external languages, such as C/C++, FORTRAN, Java, COM, and Microsoft Excel

2.1.4.1 Developing Algorithms and Applications

MATLAB provides a high-level programming language and development tools which enable you to quickly develop and analyze algorithms and applications.

The MATLAB language includes vector and matrix operations that are fundamental to solving scientific and engineering problems. This streamlines both development and execution.

With the MATLAB language, it is possible to program and develop algorithms faster than with traditional languages because it is no longer necessary to perform low-level administrative tasks, such as declaring variables, specifying data types and allocating memory. In many cases, MATLAB eliminates the need for 'for' loops. As a result, a line of MATLAB code usually replaces several lines of C or C++ code.

At the same time, MATLAB offers all the features of traditional programming languages, including arithmetic operators, control flow, data structures, data types, object-oriented programming (OOP) and debugging.

MATLAB enables you to execute commands or groups of commands one at a time, without compiling or linking, and to repeat the execution to achieve the optimal solution.

2.1.4.2 Data Visualization

All graphics functions necessary to visualize scientific and engineering data are available in MATLAB. This includes tools for two- and three-dimensional diagrams, three-dimensional volume visualization, tools to create diagrams interactively, and the ability to export using the most popular graphic formats. It is possible to customize diagrams, adding multiple axes, changing the colors of lines and markers, adding annotations, LaTeX equations and legends, and plotting paths.

Various two-dimensional graphical representations of vector data can be created, including:

- Line, area, bar and sector diagrams
- Direction and velocity diagrams
- Histograms
- Polygons and surfaces
- Dispersion bubble diagrams
- Animations

MATLAB also provides functions for displaying two-dimensional arrays, three-dimensional scalar data and three-dimensional vector data. It is possible to use these functions to visualize and understand large amounts of complex multi-dimensional data. It is also possible to define the characteristics of the diagrams, such as the orientation of the camera, perspective, lighting, light source and transparency. Three-dimensional diagramming features include:

- Surface, contour and mesh plots
- Space curves
- Cone, phase, flow and isosurface diagrams

2.1.4.3 Image Acquisition Toolbox

Image Acquisition Toolbox™ enables you to acquire images and video from cameras and frame grabbers directly into MATLAB® and Simulink®. You can detect hardware automatically, and configure hardware properties. Advanced workflows let you trigger acquisition while processing in-the-loop, perform background acquisition, and synchronize sampling across several multimodal devices. With support for multiple hardware vendors and industry standards, you can use imaging devices, ranging from inexpensive Web cameras to high-end scientific and industrial devices that meet low-light, high-speed, and other challenging requirements

Key Features

- Support for industry standards, including DCAM, Camera Link, and GigE Vision
- Support for common OS interfaces for webcams, including Direct Show, QuickTime, and video4linux2
- Support for a range of industrial and scientific hardware vendors
- Multiple acquisition modes and buffer management options
- Synchronization of multimodal acquisition devices with hardware triggering
- Image Acquisition app for rapid hardware configuration, image acquisition, and live video previewing
- Support for C code generation in Simulink

Together, MATLAB, Image Acquisition Toolbox, and Image Processing Toolbox™ (and, optionally, Computer Vision System Toolbox™) provide a complete environment for developing customized imaging solutions. You can acquire images and video, visualize data, develop processing algorithms and analysis techniques, and create user interfaces and apps. The image acquisition engine enables you to acquire frames as fast as your camera and PC can support for high speed imaging. In addition, you can use Image Acquisition Toolbox with Simulink and Computer Vision System Toolbox to model and simulate real-time embedded imaging systems.

Image Acquisition Toolbox simplifies the acquisition process by providing a consistent interface across operating systems, hardware devices, and vendors. The toolbox provides multiple ways to access hardware devices from MATLAB and Simulink: the Image Acquisition Tool, a programmatic interface in MATLAB, and a block for Simulink. Each workflow provides access to camera properties and controls while enabling you to solve different types of problems with the strengths of each environment.

2.1.4.4 Image Processing Toolbox

Image Processing Toolbox™ provides a comprehensive set of reference-standard algorithms, functions, and apps for image processing, analysis, visualization, and algorithm development. You can perform image analysis, image segmentation, image enhancement, noise reduction, geometric transformations, and image registration. Many toolbox functions support multicore processors, GPUs, and C-code generation.

Image Processing Toolbox supports a diverse set of image types, including high dynamic range, gigapixel resolution, embedded ICC profile, and tomographic. Visualization functions and apps let you explore images and videos, examine a region of pixels, adjust color and contrast, create contours or histograms, and manipulate regions of interest (ROIs). The toolbox supports workflows for processing, displaying, and navigating large images.

2.1.4.5 RobotVision toolbox

The Toolbox has always provided many functions that are useful for the study and simulation of classical arm type robotics, for example such things as kinematics, dynamics, and trajectory generation. The Toolbox is based on a very general method of representing the kinematics and dynamics of serial-link manipulators.

These parameters are encapsulated in MATLAB robot objects can be created by the user for any serial-link manipulator and a number of examples are provided for well know robots such as the Puma 560 and the Stanford arm amongst others. The Toolbox also provides functions for manipulating and converting between data types such as vectors, homogeneous transformations and unit-quaternions which are necessary to represent 3-dimensional position and orientation.

This ninth release of the Toolbox has been significantly extended to support mobile robots. For ground robots the Toolbox includes standard path planning algorithms (bug, distance transform, D*, PRM), kinodynamic planning (RRT), localization (EKF, particle filter), map building (EKF) and simultaneous localization and mapping (EKF), and a Simulink model a of non-holonomic vehicle. The Toolbox also includes a detailed Simulink model for a quadrotor flying robot.

The routines are generally written in a straightforward manner which allows for easy understanding, perhaps at the expense of computational efficiency. If you feel strongly about computational efficiency then you can always rewrite the function to be more efficient, compile the M-file using the MATLAB compiler, or create a MEX version.

The book “Robotics, Vision & Control” [13] provides a detailed discussion (600 pages, nearly 400 figures and 1000 code examples) of how to use the Toolbox functions to solve many types of problems in robotics.

2.2 Robotic Arm Kinematics

Kinematics is the branch of mechanics that studies the motion of a body or a system of bodies without consideration given to its mass or the forces acting on it. A serial-link manipulator comprises a chain of mechanical links and joints. Each joint can move its outward neighboring link with respect to its inward neighbor. One end of the chain, the base, is generally fixed and the other end is free to move in space and holds the tool or end-effector.

Each robot has multiple joints and clearly the pose of the end-effector will be a complex function of the state of each joint

2.2.1 Describing Arms joint and link structure

A serial-link manipulator comprises a set of bodies, called links, in a chain and connected by joints. Each joint has one degree of freedom, either translational (a sliding or prismatic joint) or rotational (a revolute joint). Motion of the joint changes the relative angle or position of its neighboring links

The joint structure of a robot can be described by a string such as “PRRRRR” where the J^{th}character represents the type of joint j, either Revolute or Prismatic. A systematic way of describing the geometry of a serial chain of links and joints was proposed by Denavit and Hartenberg in 1955 and is known today as Denavit-Hartenberg notation.

For a manipulator with N joints numbered from 1 to N, there are N +1 links, numbered from 0 to N. Link 0 is the base of the manipulator and link N carries the end- effector or tool. Joint j connects link j −1 to link j and therefore joint j moves link j. A link is considered a rigid body that defines the spatial relationship between two neighboring joint axes. A link can be specified by two parameters, its length a_j and its twist α_j. Joints are also described by two parameters. The link offset d_j is the distance from one link coordinate frame to the next along the axis of the joint. The joint angle θ_j is the rotation of one link with respect to the next about the joint axis.

Figure 2.1 illustrates this notation. The coordinate frame {j} is attached to the far (distal) end of link j. The axis of joint j is aligned with the z-axis. These link and joint parameters are known as Denavit-Hartenberg parameters and are summarized in Table 2.1.

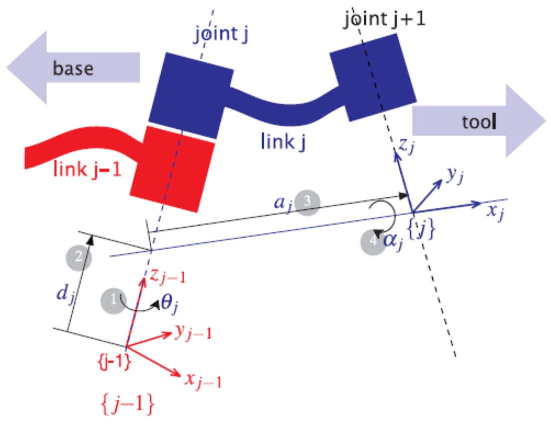
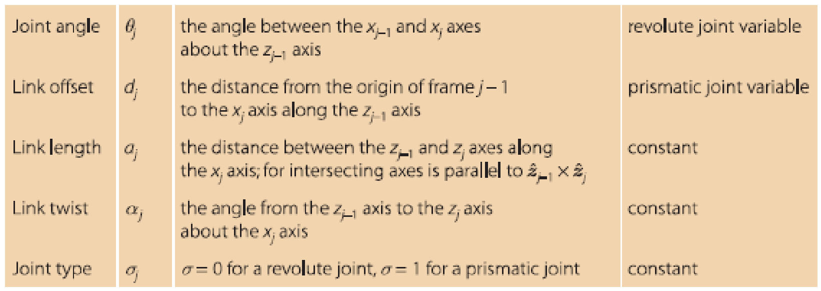


Figure 2.1: Definition of standard Denavit and Hartenberg link parameters. The colors red and blue denote all things associated with links j−1 and j respectively. The numbers in circles represent the order in which the elementary transforms are applied (Photo from: [13])

Table 2.1 Denavit-Hartenberg parameters: their physical meaning, symbol and formal definition



Joint angle	θ_j	the angle between the x_{j-1} and x_j axes about the z_{j-1} axis	revolute joint variable
Link offset	d_j	the distance from the origin of frame $j-1$ to the x_j axis along the z_{j-1} axis	prismatic joint variable
Link length	a_j	the distance between the z_{j-1} and z_j axes along the x_j axis; for intersecting axes is parallel to $\hat{z}_{j-1} \times \hat{z}_j$	constant
Link twist	α_j	the angle from the z_{j-1} axis to the z_j axis about the x_j axis	constant
Joint type	σ_j	$\sigma = 0$ for a revolute joint, $\sigma = 1$ for a prismatic joint	constant

The transformation from link coordinate frame {j − 1} to frame {j} is defined in terms of elementary rotations and can be expanded as:

$$ {}^{j-1}\mathbf{A}_j = \begin{pmatrix} \cos\theta_j & -\sin\theta_j\cos\alpha_j & \sin\theta_j\sin\alpha_j & a_j\cos\theta_j \\ \sin\theta_j & \cos\theta_j\cos\alpha_j & -\cos\theta_j\sin\alpha_j & \alpha_j\sin\theta_i \\ 0 & \sin\alpha_j & \cos\alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{...............Equation (1)} $$

The parameters α_j and a_j are always constant. For a revolute joint θ_j is the joint variable and dj is constant, while for a prismatic joint d_j is variable, θ_j is constant and $\alpha_j = 0$. In many of the formulations that follow we use generalized joint coordinates

$$ q_j = \begin{cases} \theta_j & \sigma_j = 0, \text{ for a revolute joint} \\ d_j & \sigma_j = 1, \text{ for a prismatic joint} \end{cases} \quad \text{...............Equation (2)} $$

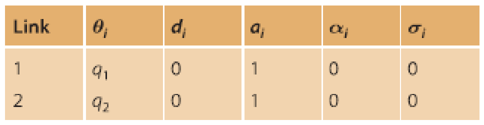
2.2.2 Forward Kinematics

Forward kinematics refers to the use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joints parameters. The forward kinematic solution can be computed for any serial-link manipulator irrespective of the number of joints or the types of joints

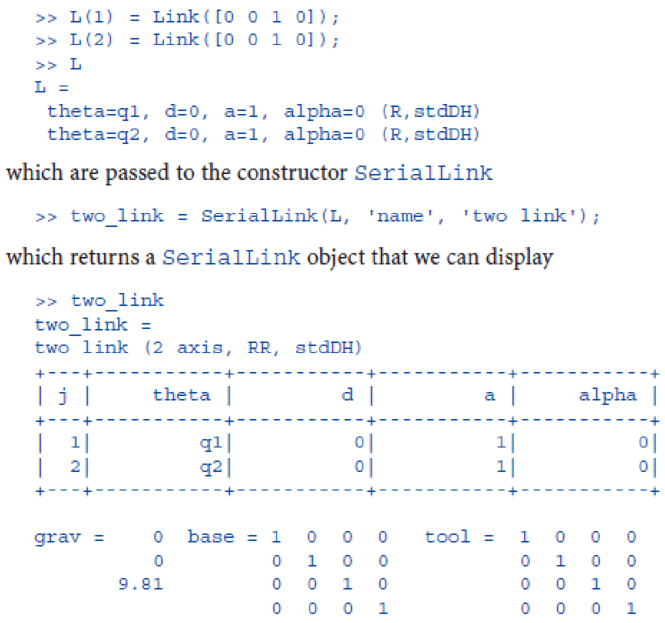
The pose of the end-effector has six degrees of freedom – three in translation and three in rotation. Therefore robot manipulators commonly have six joints or degrees of freedom to allow them to achieve an arbitrary end-effector pose.

2.2.2.1 A 2-Link Robot

The first robot that will discuss is the two-link planar manipulator shown in Figure 2.2 It has the following Denavit-Hartenberg parameters which we use to create a vector of Link objects using MATLAB Robotic ToolBox Link Command.



Link	θ_i	d_i	a_i	α_i	σ_i
1	q_1	0	1	0	0
2	q_2	0	1	0	0



```
>> L(1) = Link([0 0 1 0]);
>> L(2) = Link([0 0 1 0]);
>> L
L =
 theta=q1, d=0, a=1, alpha=0 (R,stdDH)
 theta=q2, d=0, a=1, alpha=0 (R,stdDH)
```

which are passed to the constructor `SerialLink`

```
>> two_link = SerialLink(L, 'name', 'two link');
```

which returns a `SerialLink` object that we can display

```
>> two_link
two_link =
two link (2 axis, RR, stdDH)
+---+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |
+---+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          1|          0|
|  2|         q2|          0|          1|          0|
+---+-----------+-----------+-----------+-----------+

grav =     0  base = 1  0  0  0   tool =  1  0  0  0
           0         0  1  0  0           0  1  0  0
        9.81         0  0  1  0           0  0  1  0
                     0  0  0  1           0  0  0  1
```

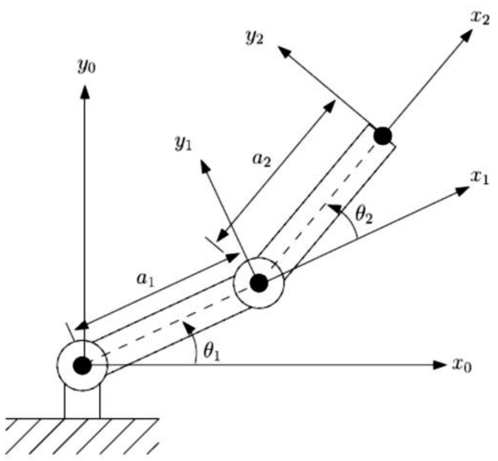


Figure 2.2 Two link Robot (Photo from: [13])

This provides a concise description of the robot. We see that it has 2 revolute joints as indicated by the structure string `'RR'`, it is defined in terms of standard Denavit-Hartenberg parameters, that gravity is acting in the default z-direction . The kinematic parameters of the link objects are also listed and the joint variables are shown as variables `q1` and `q2`. We have also assigned a name to the robot which will be shown whenever the robot is displayed graphically. The script

```
>> mdl_twolink
```

performs the above steps and defines this robot in the MATLAB® workspace, creating a `SerialLink` object named `twolink`. The `SerialLink` object is key to the operation of the Robotics Toolbox. It has a great many methods and properties which are illustrated in the rest of this part, and described in detail in the online documentation. Some simple examples are

```
>> twolink.n
ans =
       2
```

which returns the number of joints, and

```
>> links = twolink.links
L =
theta=q1, d=0, a=1, alpha=0 (R,stdDH)
theta=q2, d=0, a=1, alpha=0 (R,stdDH)
```

which returns a vector of Link objects comprising the robot. We can also make a copy of a `SerialLink` object

```
>> clone = SerialLink(twolink, 'name', 'bob')
clone =
bob (2 axis, RR, stdDH)
+---+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |
+---+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          1|          0|
|  2|         q2|          0|          1|          0|
+---+-----------+-----------+-----------+-----------+

grav =     0  base = 1   0   0   0    tool =  1   0   0   0
           0           0   1   0   0          0   1   0   0
        9.81           0   0   1   0          0   0   1   0
                       0   0   0   1          0   0   0   1
```

which has the name `'bob'`.

Now we can put the robot arm to work. The forward kinematics are computed using the `fkine` method. For the case where q1=q2= 0

```
>> twolink.fkine([0  0])
ans =
      1      0      0      2
      0      1      0      0
      0      0      1      0
      0      0      0      1
```

the method returns the homogenous transform that represents the pose of the second link coordinate frame of the robot, T2. For a different configuration the tool pose is

```
>> twolink.fkine([pi/4  -pi/4])
ans =
    1.0000           0           0      1.7071
         0      1.0000           0      0.7071
         0           0      1.0000           0
         0           0           0      1.0000
```

By convention, joint coordinates with the Toolbox are row vectors.

The robot can be visualized graphically by the next commands as shown in Figure 2.3

```
>> twolink.plot([0  0])
>> twolink.plot([pi/4  -pi/4])
```

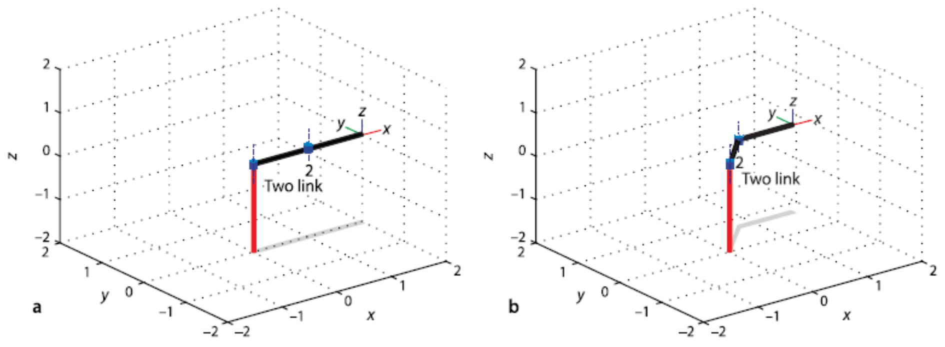


Figure 2.3: The two-link robot in two different poses, (a) the pose (0, 0); (b) the pose$\left(\frac{\pi}{4}, -\frac{\pi}{4}\right)$. Note the graphical details. Revolute joints are indicated by cylinders and the joint axes are shown as line segments. The final-link coordinate frame and a shadow on the ground are also shown (Photo from: [13])

2.2.2.2 A 6-Axis Robot

Truly useful robots have a task space $\mathcal{T} \subset SE(3)$ enabling arbitrary position and attitude of the end-effector – the task space has six spatial degrees of freedom: three translational and three rotational. This requires a robot with a configuration space $\mathcal{C} \subset \mathbb{R}^6$ which can be achieved by a robot with six joints. In this section we will use the Puma 560 as an example of the class of all-revolute six-axis robot manipulators. We define an instance of a Puma 560 robot using the script

```
>> mdl_puma560
```

which creates a SerialLink object, p560, in the workspace

```
>> p560
p560 =
Puma 560 (6 axis, RRRRRR, stdDH)
 Unimation; viscous friction; params of 8/95;
+---+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |
+---+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|      1.571|
|  2|         q2|          0|     0.4318|          0|
|  3|         q3|       0.15|     0.0203|     -1.571|
|  4|         q4|     0.4318|          0|      1.571|
|  5|         q5|          0|          0|     -1.571|
|  6|         q6|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+

grav =     0  base = 1  0  0  0   tool =  1  0  0  0
           0         0  1  0  0           0  1  0  0
        9.81         0  0  1  0           0  0  1  0
                     0  0  0  1           0  0  0  1
```

Note that a_j and d_j are in SI units which means that the translational part of the forward kinematics will also have SI units. The script `mdl_puma560` also creates a number of joint coordinate vectors in the workspace which represent the robot in some canonic configurations:

`qz`	(0, 0, 0, 0, 0, 0)	zero angle
`qr`	$\left(0\,,\ \frac{\pi}{2}\,,-\,\frac{\pi}{2}\,,0\,,0\,,0\right)$	ready, the arm is straight and vertical
`qs`	$\left(0\,,\ 0\,,-\,\frac{\pi}{2}\,,0\,,0\,,0\right)$	stretch, the arm is straight and horizontal
`qn`	$\left(0\,,\ \frac{\pi}{4}\,,\ -\pi\,,0\,,\ \frac{\pi}{4}\,,0\right)$	nominal, the arm is in a dextrous working pose

and these are shown graphically in Figure 2.4. These plots are generated using the plot method, for example

```
>> p560.plot(qz)
```

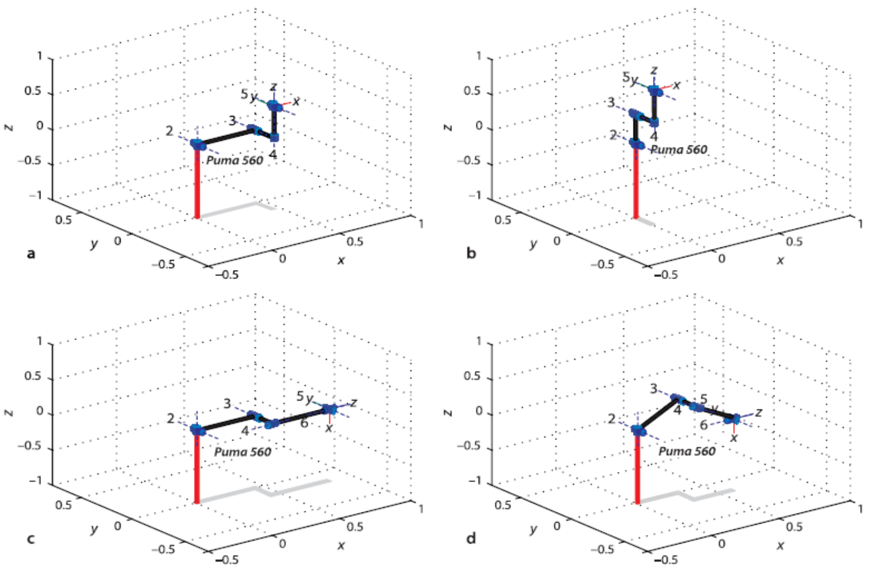


Figure 2.4: The Puma robot in 4 different poses. (a) Zero angle; (b) ready pose; (c) stretch;(d) nominal (Photo from: [13])

Forward kinematics can be computed as before

```
>> p560.fkine(qz)
ans =
    1.0000         0         0    0.4521
         0    1.0000         0   -0.1500
         0         0    1.0000    0.4318
         0         0         0    1.0000
```

which returns the homogeneous transformation corresponding to the end-effector pose T_6. The origin of this frame, the link 6 coordinate frame $\{6\}$, is defined as the point of intersection of the axes of the last 3 joints – physically this point is inside the robot's wrist mechanism.

2.2.3 Inverse Kinematics

We have shown how to determine the pose of the end-effector given the joint coordinates and optional tool and base transforms. A problem of real practical interest is the inverse problem: given the desired pose of the end-effector ξ_E what are the required joint coordinates? For example, if we know the Cartesian pose of an object, what joint coordinates does the robot need in order to reach it? This is the inverse kinematics problem which is written in functional form as

$$q = \kappa^{-1}(\xi)$$

In general this function is not unique and for some classes of manipulator no closed-form solution exists, necessitating a numerical solution.

2.2.4 Trajectories

One of the most common requirements in robotics is to move the end-effector smoothly from pose A to pose B. we will discuss two approaches to generating such trajectories: straight lines in joint space and straight lines in Cartesian space. These are known respectively as joint-space and Cartesian motion.

2.2.5 Determining Denavit-Hartenberg Parameters

The classical method of determining Denavit-Hartenberg parameters is to systematically assign a coordinate frame to each link. The link frames for the Puma robot using the standard Denavit-Hartenberg formalism are shown in Figure 2.6(a). However there are strong constraints on placing each frame since joint rotation must be about the z-axis and the link displacement must be in the x-direction. This in turn imposes constraints on the placement of the coordinate frames for the base and the end-effector, and ultimately dictates the zero-angle pose just discussed. Determining the Denavit-Hartenberg parameters and link coordinate frames for a completely new mechanism is therefore more difficult than it should be – even for an experienced roboticist.

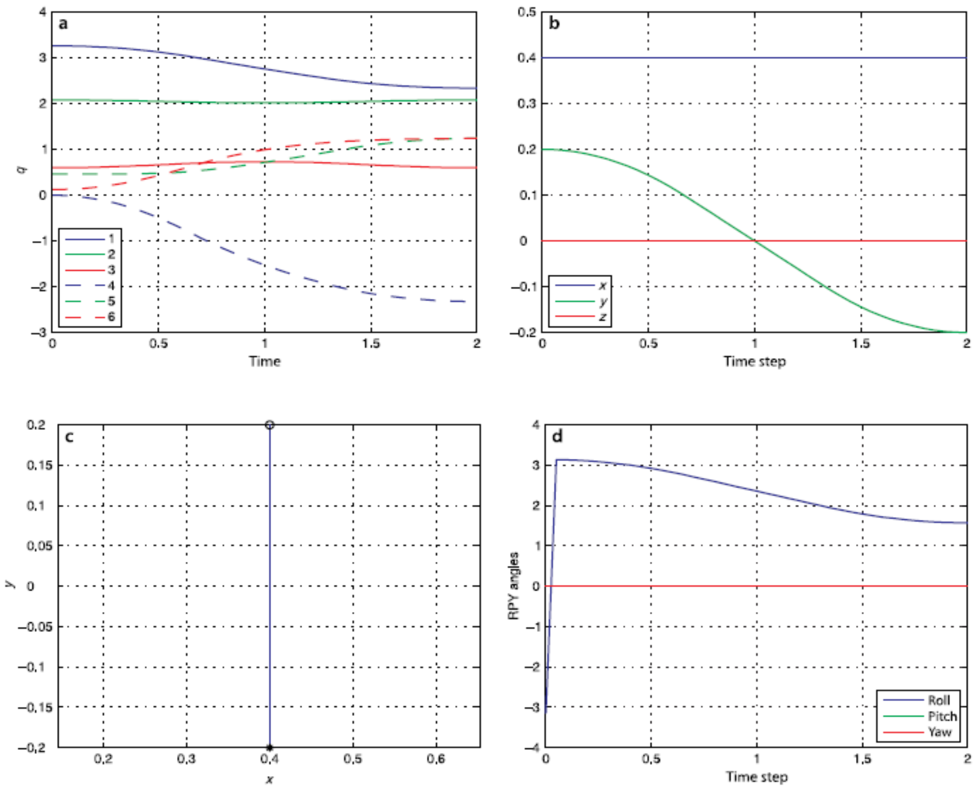


Figure 2.5: Cartesian motion. (a) Joint coordinates versus time; (b) Cartesian position versus time; (c) Cartesian position locus in the xy-plane; (d) roll-pitch-yaw angles versus time (Photo from: [13])

An alternative approach, supported by the Toolbox, is to simply describe the manipulator as a series of elementary translations and rotations from the base to the tip of the end-effector. Some of the elementary operations are constants such as translations that represent link lengths or offsets, and some are functions of the generalized joint coordinates q_i. Unlike the conventional approach we impose no constraints on the axes about which these rotations or translations can occur.

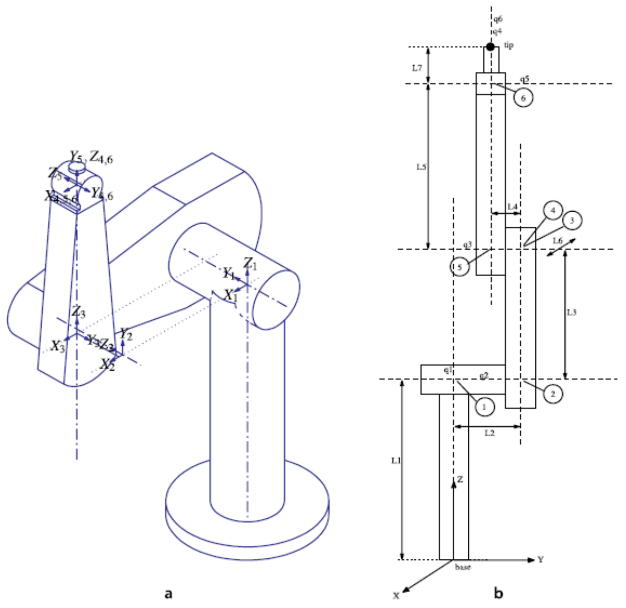


Figure 2.6: Puma 560 robot coordinate frames. (a) Standard Denavit-Hartenberg link coordinate frames for Puma in the zeroangle pose ; (b) alternative approach showing the sequence of elementary transforms from base to tip. Rotations are about the axes shown as dashed lines (Photo from: [13])

2.3 Visual Servoing

Visual servoing is the name given to the control techniques that use image data as input. It is a very large field of research and to build a system like that, it is necessary to join knowledge from many areas of research like robot modeling, control theory, computer vision and others, [2]. The main advantage of including vision in robotic systems is that it improves the flexibility and accuracy of these. In a visual servoing system, the robot should be able to extract from the image, all the environment's features which it needs to execute its task. There are two main approaches in visual servoing: image based visual servoing and position based visual servoing. Beyond these, there are also the hybrid systems which were developed to try to embody the best features of both approaches.

The two most common branches of application for visual servoing are: positioning of a robot in a desired position (typically with the goal of grasping some object) and following targets in movement. This thesis just considers the first one.

2.3.1 Background

The typical structure of a control system based in image as input is exposed in Figure 2.7

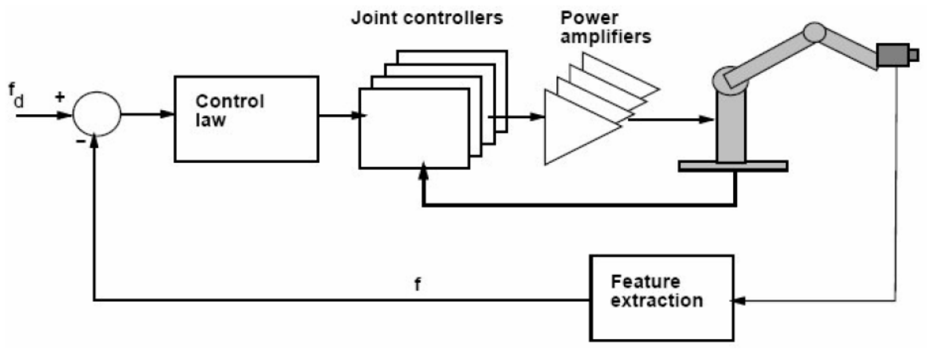


Figure 2.7: Control scheme of a visual servoing system; image based visual servoing using the eye-in-hand approach (Photo from: [9]).

The goal of a visual servoing system is to move a robot from one initial to one desired position, and keep it there using always a close-loop control scheme. In the beginning of the process, is provided to the system one image of the target with the camera in the desired position. Then, in the starting position, the camera obtains an image of the target. From that image, the necessary features, f, are extracted and compared with the desired ones extracted from the desired image,f_d; the error obtained from the difference between the desired and the current image features is used to compute the control law. The control signal is sent to the joints and a new image, with the camera in the new position, is taken. This is done until could be possible reducing to zero the error between the image features in the current and in the desired position.

Based on the image, it is possible to estimate parameters like the object's pose or the camera displacement and rotation. For that, the main challenge to solve is the image matching; how to know which point corresponds to which in two images. To solve the association problem, [30], many aspects need to be taken in account: whether the scene is textured, whether reflections or occlusion may occur, whether there are several objects in the scene, whether those are planar of curved. All these possibilities can make a big difference in the successes of the task, however, its study is out of the scope of this thesis. In the present work the target is assumed as perfectly identified.

The matching algorithms are basically divided in two approaches: correlation techniques (also called area based methods) and techniques based on object features.The minimum sum of square differences method is used in the correlations methods, which tries to minimize the differences between the two images in a specific image window. It uses the following definition:

$$\min\left\{SSD(u,v) = \sum_{n}\sum_{m}\left[I(u+m, v+n) - T(u,v)\right]^2\right\} \quad \text{……………Equation (3)}$$

In which I (u,v) is the current image, T(u,v) the desired image and (m, n) a displacement along

the (u,v) pixels directions

In the case of techniques based on object features, those features are extracted from the images and then some similarity criterions are applied. The features based methods tend to work well when manmade objects, which have not a complex shape, are tracked. If it is desired to track a specific pattern, then the area methods have a better performance. The main problem of the area based method is that they only work well when the camera is almost perpendicular with the target plane, not performing well when the camera is too much inclined. Another problem of these area based methods is that their performance depends on the light conditions, which is a problem for environments that use natural light. If some occlusion in the scene can happen, the features methods are again more robust. In all cases, to get successful results it is necessary that both images are taken close. A big displacement between both images normally leads to loss of their correspondence. The area methods are not taken into account in this thesis and have been less used for generic applications.

For techniques based on object features, the first task to do, after an image is acquired, is to extract the information from it, image processing. That information can be points, lines, ellipses, differences in the texture, which basically means edge detection. This can be done using several methods; one possibility is to use the Laplacian operator, which detects changes in the grey levels of the image. This operator consists in convolving the image with the Laplacian mask matrix (Figure 2.8 (a). An example of the application of this method can be seen in Figure 2.8.

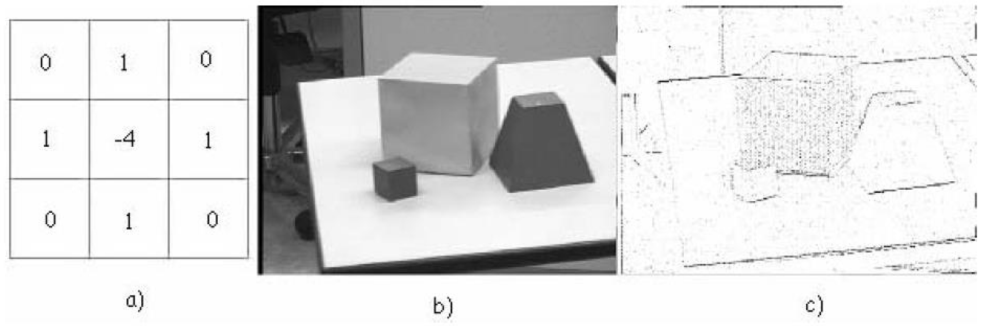


Figure 2.8: Application example of the Laplacian operator to extract edges: (a) Laplacian mask; (b) Original image; (c) Image after had been convolved with the Laplacian mask (Photo from: [9])

The main problem of this approach is that is very sensible to the noise. As can be seen in Figure 2.8 (c), there are many isolated points (mainly in the biggest cube) which do not belong to any edge; beyond that if there is some occlusion in a part of the image, the edges do not appear connected. So it is necessary to define, which points correspond to real edges. For that, the starting point is an image like in Figure 2.8 (c), in which are defined the points which were identified as possible belonging to an edge. A way to search all the straight lines present in the image is finding all lines defined by every pair of points identified as a possibly belonging to an edge (Figure 2.8

(c)); it is then possible infer that the lines that appear represented more times correspond to the real edges. However, this algorithm it is not viable due to its high computational complexity. So instead of this, a better choice is using the Hough Transform which has the advantage that can be extended to detect curves.

It is possible to define several image feature parameters. An image feature parameter is a quantity value that can be extracted from each image. A good image feature is one that can be located unambiguously in different views of the scene. The choice of the image features (primitives) to use depends of the target and the image conditions. In [27] are given some hints to how to choose the most efficient features for a specific task. Only after had get those primitives, it is possible to start the task of interpretation of the surrounding robot's environment.

The extraction of features from the image can be done in a direct or indirect way. In the direct approach the variables to control lie in the 2D image plane coordinates, image based visual servoing (Figure 2.7). In the indirect approach it is necessary to extract the relevant features of the object from the image plane, and then, use a 3D model of the camera, and the target, to compute the position of the target and the camera in the world, position based visual servoing. In this second case, an extra block should be added in Figure 2.7 for pose estimation after the feature extraction block in the feedback loop.

In pose estimation, the position and orientation of the target in relation to the camera is computed. This is done from the object image and the a-priory knowledge it has of the object. There are several ways to build a model of the tracked target, which depend mainly of the object features. For objects with a well-defined and geometric shape the most common approach is use a wire-frame representation to define it; Figure 2.9 (a) ([28]), gives one example of this representation. An overview of modeling elementary polygons is given by Vincze in [29].

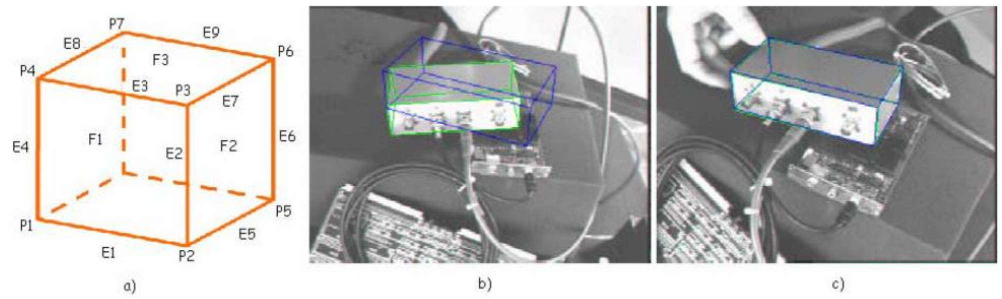


Figure 2.9: Object model for position based visual servoing. (a) Possible representation of the object tracked (P – points, E – edges, F - polygons); (b) Projection of the object model in a not final position; (c) Projection of the object model in the final position. (Photo from: [9])

To estimate the position of the object in relation to the camera, the object model is projected in the image (only the visible edges), Figure 2.9 (b); then, some algorithms try to match the object edges obtained from the current image with the object model projection. Then, from the object model, it is possible to infer the spatial position of the object. Some of these matching algorithms are presented in [30] and [31]. In the desired, position the object edges and the object model projection should coincide, Figure 2.9 (c). A survey of model-based approaches can be found in [32].

The goal of pose estimation is to have a fast, accurate and robust algorithm. However, normally it is not possible to get all these specifications at the same time. An analysis of the tradeoff between speed, accuracy and robustness in position based visual servoing methods was performed by Wilson in [33].

The visual servoing control law structure is almost the same for all the methods, changing basically the features used for tracking and the quantity of object a priori knowledge used. The definitions presented in this chapter are generic for all the methods and their notation is subsequently presented. The feature vector, s , is defined by the list of features, extracted directly or indirectly from the image, used in the control law. The features error, or task vector, e , is the difference between the current features vector and the desired features vector

$$e = s_{current} - s^{*}_{desired} \qquad \text{..............Equation (4)}$$

For a static target, it is obtained the equality current $\dot{e} = \dot{s}_{current}$

To ensure the zeroing of the features errors in the task space, it is imposed a restriction in the control law defined by the following first order equation

$$\dot{e} = -\lambda \cdot e \qquad \text{..............Equation (5)}$$

In visual servoing is necessary to make a transformation between the Cartesian space and the feature space. This is done by the image Jacobian, J , also called interaction matrix, L_s , which relates both dimensions

$$J_q = \frac{\partial e}{\partial q} \qquad \text{..............Equation (6)}$$

The vector $q_{[1\times16]}$ represents the position of the joints and ∂q their velocities.

Inverting the relation

$$\dot{e} = \dot{s} = J \cdot v = L_s \cdot v \qquad \text{..............Equation (7)}$$

it is possible to obtain the control law

$$v = -\lambda \cdot J^{+} \cdot \left(s_{current} - s^{*}\right) \qquad \text{..............Equation (8)}$$

The difference between Equation 6 and Equation 7 is that the first is expressed in the joint space, (v,Jq) , and the second in the Cartesian space, (v, J). The velocity, v , is a [1x6] vector, in which the first three components are associated with the translation, and the last three with the rotation, $v = \lfloor v_x \quad v_y \quad v_z \quad \omega_x \quad \omega_y \quad \omega_z \rfloor$. Typically J is a not square matrix, so in Equation 8, it is necessary to use the pseudo-inverse of J .

The velocity obtained in Equation 8 refers to the position of the camera, although normally, for example to pick up an object, it is desired control the position of the end-effector of the robotic arm instead. For that, it is necessary to change the reference point from the camera to the end-effector. This can be done with a transformation matrix, ${}^{e}V_c$, defined by

$$ {}^{e}V_c = \begin{bmatrix} {}^{e}R_c & \left[{}^{e}t_c\right]_x \cdot {}^{e}R_c \\ 0_3 & {}^{e}R_c \end{bmatrix} \qquad \text{...............Equation (9)} $$

Here, ${}^{e}R_c$ and ${}^{e}t_c$ are the rotation and translation between the camera frame and the end-effector frame, and $[\,{}^{e}t_c\,]_x$ is the anti-symmetric matrix associated to the vector ${}^{e}t_c$. The estimation of this matrix can be done offline

To transform the Cartesian velocities, v , in joint velocities, q , it is necessary to use the robot Jacobian, J_R

$$ v = J_R \cdot \dot{q} \qquad \text{..............Equation (10)} $$

In this sense, the image Jacobian of the end-effector, in the joint space, is given by

$$ J_q = J \cdot {}^{e}V_c \cdot J_R \qquad \text{..............Equation (11)} $$

The control law given in the joint coordinates and in the end-effector frame is given by

$$ \dot{q} = -\lambda \cdot J_q^{+} \cdot \left(s_{current} - s^{*}\right) \qquad \text{...............Equation (12)} $$

In visual servoing have been used a proportional controller, which it is enough to solve the task. The problem in the visual servoing are not related with the choice of the control (proportional, PD, PID) but with the other problems cited along this thesis.

2.3.2 Evaluated Methods

Here we will analyzed and evaluated five visual servoing methods. They are the position based visual servoing (PBVS), [2], the image based visual servoing (IBVS), [2], the 2.5D visual servoing (2.5D), [18], the Corke & Hutchinson visual servoing (CH), [34], and the Kyrki and Kragic Shortest Path (SP), [35].

2.3.3 Position Based Visual Servoing

In position based visual servoing, the features are extracted indirectly from the image; it is also called 3D visual servoing; the data is transposed from the 2D image plane to 3D Cartesian space. The image is used to localize the object in the world.

PBVS has some disadvantages: it is necessary to have a thorough knowledge about the target to build a specific model of it, which is only possible to apply to objects with a specific shape; this also reduces the applicability of the algorithm; other disadvantage is that, due to the control calculus are done in the Cartesian space, the system requires an accuracy calibration; another problem is that this computation, in the Cartesian space, also requires a high computational effort, which reduces the applicability of the algorithm in a real time system. Beyond this, there is not any control over the image plane, as could be seen in the end of current chapter, which means that the target might leave the camera's field of view. The stability analysis in this method is also hard to do.

On the other hand the main advantages of PBVS are the accuracy and robustness of its performance.

In PBVS, the feature vector is defined as

$$s = \begin{bmatrix} {}^{c}t_{c*} \\ {}^{c}u_{c*}\theta_{3x1} \end{bmatrix} \quad \text{...............Equation (13)}$$

${}^{c}t_{c*}$ and ${}^{c}u_{c*}\theta_{3x1}$ represent the translation and rotation from the initial to the desired position; in this last feature, ${}^{c}u_{c*}$ are the rotation axis and θ_{3x1} are the rotation angles obtained from the rotation matrix. These features can be extracted from the transformation coordinates from the current camera's position to the desired, ${}^{c}T_{c*}$. That is obtained from the composition of the transformation coordinates from the current position to the target with the desired position to it

$${}^{c}T_{c*} = {}^{c}T_{o}\,{}^{o}T_{c*} = \begin{bmatrix} {}^{c}R_{o} & {}^{c}t_{o} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^{o}R_{c*} & {}^{o}t_{c*} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^{c}R_{o}\,{}^{o}R_{c*} & {}^{c}R_{o}\,{}^{o}t_{c*} + {}^{c}t_{o} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^{c}R_{c*} & {}^{c}t_{c*} \\ 0 & 1 \end{bmatrix} \quad \text{...............Equation (14)}$$

In PBVS, the translation and rotation of the camera (end-effector) are computed separately and are given by

$${}^{c}\dot{t}_{c*} = \begin{bmatrix} I_3 & 0_3 \end{bmatrix} \cdot v \quad \text{...............Equation (15)}$$

$${}^{c}\dot{u}_{c*}\theta_{3x1} = \begin{bmatrix} 0_3 & L_w \end{bmatrix} \cdot v \quad \text{...............Equation (16)}$$

With

$$L_w({}^{c}\dot{u}_{c*},\theta_{3x1}) = I_3 - \frac{\theta}{2}\cdot[u]_x + \left(1 - \frac{\sin c(\theta)}{\sin c^2\left(\frac{\theta}{2}\right)}\right)\cdot[u]_x^2$$

……………Equation (17)

$$\sin c(\theta) = \begin{cases} 1 & \theta = 0 \\ \frac{\sin(\theta)}{\theta} & \theta \neq 0 \end{cases}$$

Then, the position based Jacobian can be defined as

$$\dot{s} = \begin{bmatrix} {}^{c}\dot{t}_{c*} \\ {}^{c}\dot{u}_{c*}\theta_{3x1} \end{bmatrix} = \begin{bmatrix} I_3 & 0_3 \\ 0_3 & L_w \end{bmatrix}\cdot v = \begin{bmatrix} I_3 & 0_3 \\ 0_3 & I_3 \end{bmatrix}\cdot v = J_{3D}\cdot v$$ ……………Equation (18)

Because $L_w^{-1}(u,\theta)$ can be approximate by the identity matrix.

It is not strange that the image Jacobian is an identity matrix because both variables related by it, $(\dot{s}, v)$are defined in the Cartesian space.

And then, the respective control law is

……………Equation (19)

$$v = -\lambda \cdot J_{3D}^{-1}\cdot \dot{s}$$

2.3.4 Image Based Visual Servoing

In image based robot control, the target features are extracted directly from the image, without any image interpretation. The features obtained from the object are primitives like ellipsis, lines, points or moments. This control architecture does not need to make a 3D reconstruction of the target. This allows a reduction of the processing time of the algorithm and increases the sample rate which also reduces some computer vision problems. How to obtain the interaction model of IBVS is now presented.

Defining a generic point in the 3D space as:

$$\bar{x} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$$ ……………Equation (20)

The velocity of that point in relation of a referential in movement with velocity v is

$$\dot{\bar{x}} = \begin{bmatrix} -I_3 & [\bar{x}]_x \end{bmatrix}\cdot v_{[1x6]} = \begin{bmatrix} \dot{X} & \dot{Y} & \dot{Z} \end{bmatrix}^T$$ ……………Equation (21)

In which $[\bar{x}]_x$ is the anti-symmetric matrix associated to the vector $\bar{x}$. A point in the image is defined in metric coordinates as

$$m = \begin{bmatrix} x & y \end{bmatrix}^T = \begin{bmatrix} \frac{X}{Z} & \frac{Y}{Z} \end{bmatrix}^T$$ ……………Equation (22)

Then, the derivation of this point in order to the time is

$$\dot{m} = \begin{bmatrix} \dot{x} & \dot{y} \end{bmatrix} = \begin{bmatrix} \frac{\dot{X} \cdot Z - X \cdot \dot{Z}}{Z^2} \\ \frac{\dot{Y} \cdot Z - Y \cdot \dot{Z}}{Z^2} \end{bmatrix} = \frac{1}{Z} \cdot \begin{bmatrix} 1 & 0 & -\frac{X}{Z} \\ 0 & 1 & -\frac{Y}{Z} \end{bmatrix} \cdot \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix}$$ ……………Equation (23)

……………Equation (24)

$$\dot{m} = -J_v(m) \cdot \dot{\bar{x}}$$

With

$$J_v(m) = \frac{1}{Z} \cdot \begin{bmatrix} -1 & 0 & \frac{X}{Z} \\ 0 & -1 & \frac{Y}{Z} \end{bmatrix}$$ ……………Equation (25)

Using the Equations 21 and 24, it is possible to obtain the relation

$$\dot{m} = -J_v(m) \cdot \begin{bmatrix} -I_3 & [\bar{x}]_x \end{bmatrix} \cdot v$$ ……………Equation (26)

$$\dot{m} = \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{X}{Z^2} & x \cdot y & -(1 + x^2) & y \\ 0 & -\frac{1}{Z} & \frac{Y}{Z^2} & 1 + y^2 & -x \cdot y & -x \end{bmatrix} \cdot v$$ ……………Equation (27)

In metric images coordinates:

$$\dot{m} = \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{x}{Z} & x \cdot y & -\left(1+x^2\right) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & 1+y^2 & -x \cdot y & -x \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$ ……………Equation (28)

$$\dot{m} = -J \cdot v = \begin{bmatrix} \frac{1}{Z} & 0 & -\frac{x}{Z} & -x \cdot y & \left(1+x^2\right) & -y \\ 0 & \frac{1}{Z} & -\frac{y}{Z} & -\left(1+y^2\right) & x \cdot y & x \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$ ……………Equation (29)

Then, applying this approach to the IBVS, the feature vector, s , is defined as

$$s = \begin{bmatrix} x \\ y \end{bmatrix}$$ ……………Equation (30)

The variables x and y are the point's image plane coordinates and the error vector is

$$\dot{s} = \begin{bmatrix} x - x^* \\ y - y^* \end{bmatrix}$$ ……………Equation (31)

The relation between the features error and the velocity to send to the robot is

$$\dot{s} = J \cdot v$$ ……………Equation (32)

$$v = -\lambda \cdot J_{2D}^{+} \cdot \dot{s}$$ ……………Equation (33)

The Jacobian presented in the Equation 29 has the dimension of [2 x 6]. This is the Jacobian size to track one single point. For *n* points, the Jacobian dimension would be [*n* 2 x 6]. It is possible to compute the image Jacobian defined in Equation 29, in some different ways. The first possibility is to use at each iteration the real values of x , y and Z . The last parameter is hard to be measured only from the visual data information, although when any other information is provided, one way to solve this problem is to use a constant likelihood value, which normally performs correctly.

Regardless of providing the most efficient computation of the Jacobian, using the real values at each iteration requires a great computation effort. Another way to compute the image Jacobian is to use a constant Jacobian; this can be computed with base on the information of the initial position which can cause large and unnecessary movements of the camera in its path; or with base on the desired position, which sometime, can make the algorithm diverge; typically using the average value of both values $((L_s^i + L_s^*)/2)$ results better than using only the information based in the initial or in the desired position. Another possibility is to use learning method to estimate the Jacobian (on-line or ofline), [42]. In this project the image Jacobian is always computed based on the real values of the image pixels and depth value, at each iteration.

To ensure the convergence of the control law, it is necessary that

$$J_{2D} \cdot J_{2D}^{+} > 0$$

This depends of the depth values. This means that for a Jacobian updated at each iteration, only a local convergence can be ensured. This local convergence is one of the problems of IBVS. The second main problem is the difficulty of this method in managing rotations around the optical axis. As can be seen in the experimental results of this thesis, each point in the image plane tries to make a straight line into the desired position. Because of that, in the case of rotations around the optical axis, the camera needs to retreat, and the correspondent displacement is as high as the rotation is close to 180º; for this value, the camera theoretically needs to retreat until the infinite – this problem in IBVS is known as Chaumette Conundrum. A global stability analysis of IBVS and also PBVS was investigated by Chaumette in [36].

To compute the control law for image based visual servoing, it is necessary to have at least three non-collinear points. However, doing it with a higher number of points increases its robustness.

In general, the accuracy in the image-based methods for static positioning is less sensitive to calibration than position based methods due to does not use the homography to compute the control law.

2.3.5 2 ½ D Visual Servoing

The 2.5D visual servoing method was presented by Malis [18] in 1999 and was developed with the goal of joining the best skills of the two main visual servoing approaches (IBVS and PBVS).

This hybrid method does not need any 3D model of the object and ensures the convergence of the control law in the whole task space. This method itself does not ensure that the target does not leave

the camera's view field, which could happen mainly when the initial position of the camera is very far away from the target; however, this problem can be addressed including some additional techniques in the control law.

In the 2.5D, the task vector is defined as

$$e_{[6x1]} = \left(x, y, \log(\frac{Z}{Z^*}), {}^{c}u_{c*}\theta_{3x1} \right). \qquad \text{...............Equation (34)}$$

In which, x and y are the metric coordinates of a point, Z the current distance of the camera to that point and u_{3x1} and $\theta_{3x1}$are the axis of the rotation and the angles that the camera needs to rotate Figure 2.10.

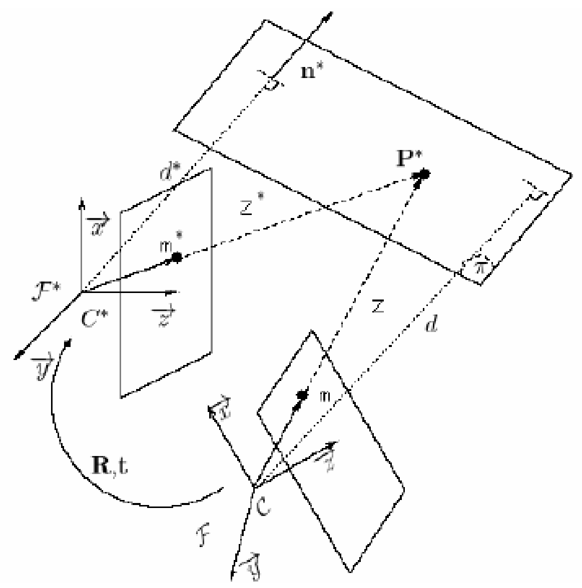


Figure 2.10 Variables used in 2.5D visual servoing control law (Photo from: [9]).

The velocity of one point, which lies on the target plane, π, relatively to one referential in movement, with velocity, v, is given by

$$\dot{\bar{x}} = \begin{bmatrix} -I_3 & [\bar{x}]_x \end{bmatrix} \cdot v \qquad \text{...............Equation (35)}$$

In which $[\bar{x}]_x$ is the ant-symmetric matrix associated to the vector $\bar{x}$.

To control the orientation the , ${}^{c}u_{c*}\ \theta_{3x1}$ approach is used, which has the advantage of having no singularities in whole workspace. The time derivative of ${}^{c}u_{c*}\ \theta_{3x1}$ (v) is:

$${}^{c}\dot{u}_{c*}\theta_{3x1} = \begin{bmatrix} 0 & L_w \end{bmatrix} \cdot v \qquad \text{...............Equation (36)}$$

In which w L is defined as

$$L_w({}^{c}u_{c*}, \theta_{3x1}) = I_3 - \frac{\theta}{2} \cdot [u]_x + \left(1 - \frac{\sin c(\theta)}{\sin c^2\left(\frac{\theta}{2}\right)} \right) \cdot [u]_x^2 \qquad \text{...............Equation (37)}$$

The extended image coordinates of a point in the target can be defined as

$$m_e = [x \quad y \quad z]^T = \left[\frac{X}{Z} \quad \frac{Y}{Z} \quad \ln(Z)\right]^T$$ ……………Equation (38)

With the depth defined as z = log(Z) . The time derivative of these coordinates is

$$\dot{m}_e = [\dot{x} \quad \dot{y} \quad \dot{z}]^T = \begin{bmatrix} \frac{\dot{X}\cdot Z - X\cdot \dot{Z}}{Z^2} \\ \frac{\dot{Y}\cdot Z - Y\cdot \dot{Z}}{Z^2} \\ \frac{Z}{Z} \end{bmatrix} = \frac{1}{Z}\cdot \begin{bmatrix} 1 & 0 & -\frac{X}{Z} \\ 0 & 1 & -\frac{Y}{Z} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = -\frac{1}{d^*}\cdot L_v \cdot \dot{x}$$
……………Equation (39)

With

$$L_v = \frac{1}{\rho}\cdot \begin{bmatrix} -1 & 0 & x \\ 0 & -1 & y \\ 0 & 0 & -1 \end{bmatrix}$$ ……………Equation (40)

$$\rho = \frac{Z}{d^*}$$ ……………Equation (41)

In which d* represents the distance to the target in the desired position, see Figure 2.9.

Then, using the Equation 35, the time derivative of the coordinates is given by

……………Equation (42)
$$\dot{m}_e = \frac{1}{d^*}\cdot L_v \cdot [-I_3 \quad [x]_x]\cdot v$$

$$\dot{m}_e = \left[\frac{1}{d^*}L_v \quad L_{v,w}\right] = \begin{bmatrix} \frac{1}{d^*}\cdot \begin{pmatrix} -1 & 0 & x \\ 0 & -1 & y \\ 0 & 0 & -1 \end{pmatrix} & \begin{matrix} xy & -(1+y^2) & y \\ (1+y^2) & -xy & -x \\ -y & x & 0 \end{matrix} \end{bmatrix}\cdot v$$ ……………Equation (43)

From Equations 36 and 43 it is possible to get the expression of the features error and the features error derivate.

$$e = \left[(m_e - m_e^*)^T \quad {}^c u_{c*} \theta_{3x1} \right]^T$$ ……………Equation (44)

$$\dot{e}_{6x1} = J_{2.5D} \cdot v_{6x1}$$ ……………Equation (45)

With

$$J_{2.5D} = \begin{bmatrix} \frac{1}{d^*} \cdot L_v & L_{(v,w)} \\ 0 & L_w \end{bmatrix}$$ ……………Equation (46)

Then, the control law is:

$$v = -\lambda \cdot J_{2.5D}^{-1} \cdot e$$ ……………Equation (47)

$J_{2.5D}$ is a upper triangle matrix which its inverse is not hard to compute and $L_w^{-1}\ (u, \theta)$ can be approximate by the identity matrix .

The control law is then defined as:

$$v = -\lambda \cdot \begin{bmatrix} \hat{d}^* \cdot L_v^{-1} & \hat{d}^* \cdot L_v^{-1} \cdot L_{(v,w)} \\ 0 & I_3 \end{bmatrix} \begin{bmatrix} m_e - m_e^* \\ {}^c u_{c*} \theta_{3x1} \end{bmatrix}$$ ……………Equation (48)

2.3.6 Corke & Hutchinson Visual Servoing

The Corke and Hutchinson method, [34], was developed to address some of the IBVS problems, mainly the camera retreat problem, which occurs for pure rotations around the camera's optical axis.

The idea of this image based partitioned method is to separate the control translation and rotation over the z-axis from the other degrees of freedom. The CH method has the drawback that its performance depends of the relative position of the features in the image.

The interaction model is defined, as in all the other methods, as

$$\dot{e} = J_{C\&H} \cdot v$$ ……………Equation (49)

In this case the error vector is defined as in IBVS

$$e = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} x - x^* \\ y - y^* \end{bmatrix}$$ ……………Equation (50)

Only one point is tracked, and (x , y) are their image plane coordinates.

In this method the translation, v_z, and the rotation, ω_z , velocities along/around z-axis are computed from the next features

2.3.6.1 Theta Feature

To drive the camera inclination around the optical axis it is used the angle theta, θ . It is defined as the angle between the horizontal pixel axis and a straight line built from two points in the image, as is shown in Figure 2.11.

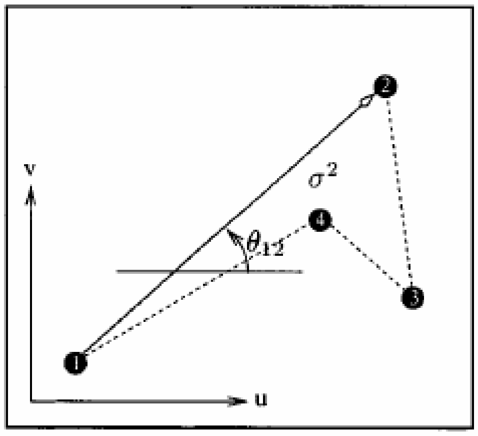


Figure 2.11: Features used in the Corke and Hutchinson visual servoing control law (Photo from: [9]).

This can be done with any pair of points, but the accuracy of the feature increases as far as the points lie on the image

2.3.6.2Sigma Feature

The control thought the z-axis is computed from the sigma-square feature, σ^2, which consists in the area of a polygon made from the image points; for that at least three not collinear points are needed.

To use a feature with the unities of a length (in pixels) instead of the area, the feature used is the square root of the area of that polygon, $= \sqrt{area}$. If the camera is too far from the target (in relation of the desired position), the area, σ^2, of the polygon is smaller and the camera needs to move forward, if the area is too big, this needs to retreat.

The translational and rotational velocities along/around the optical axis are given by:

$$\begin{bmatrix} v_z \\ \omega_z \end{bmatrix} = \begin{matrix} \lambda_{Tz}\left(\sigma^* - \sigma\right) \\ \lambda_{\omega z}\left(\theta_{ij}^* - \theta_{ij}\right) \end{matrix} \qquad \text{...............Equation (51)}$$

λ_{Tz} and $\lambda_{\omega z}$ are gain factors to adapt the speed of conversion along the associated directions. Then, it is possible to write the Equation 49 as

$$\dot{e} = -J_{xy} \cdot v_{xy} - J_z \cdot v_z \qquad \text{...............Equation (52)}$$

Where v_{xy} and v_z are the velocities associated to the respective indexes:

$v_{xy} = [v_x \; v_y \; \omega_x \; \omega_y]$, $v_z = [v_z \; \omega_z]$, and J_{xy} and J_z are the colons of the image based visual servoing Jacobian associated with the respective indexes (Equation 29)

$$J_{xy} = \begin{bmatrix} -\frac{1}{Z} & 0 & x \cdot y & -(1+x^2) \\ 0 & -\frac{1}{Z} & 1+y^2 & -x \cdot y \end{bmatrix} \quad \text{...............Equation (53)}$$

$$J_z = \begin{bmatrix} \frac{x}{Z} & y \\ \frac{y}{Z} & -x \end{bmatrix} \quad \text{...............Equation (54)}$$

Then, the Equation 52 can be written in order of v_{xy}:

$$v_{xy} = -[J_{xy}^{+}]_{4x2} \left\{ \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} \frac{x}{Z} & y \\ \frac{y}{Z} & -x \end{bmatrix} \cdot \begin{bmatrix} \dot{\sigma} \\ \dot{\theta} \end{bmatrix} \right\} \quad \text{...............Equation (55)}$$

In the paper where this method was presented, [34], few tests were shown. This technique performed perfectly for pure rotation around the optical axis. For generic motions, this method performed little better than IBVS, despite of the excursion of the features in the image plane are more complex than in IBVS. However these tests did not include high rotations around an axis coplanar with the target plane, which could make a big distortion in the image features. Those tests were also performed under perfect condition without a presence of any extern perturbation like noise or calibration errors.

2.3.7 Shortest Path Visual Servoing

The shortest path visual servoing was presented by Kyrki and Kragic, [35], with the goal of addressing two typical problems in the visual servoing: the possibility of the target leaving the camera's field of view and the risk of the joints working too close to their limits. This method was developed to the camera presents one straight line trajectory which goes from the initial to the desired position. This approach also solves the problem of the uncertain in the trajectory. The uncertainty of the trajectory occurs in several methods and in some cases can put their success in risk.

This method can be implemented as a 3D visual servoing method or a hybrid method. Here the mathematical description of the first is presented, which is implemented in this project. The difference between both approaches is just the way that the depth information is obtained. In the case of the hybrid approach, the depth estimation can be done as in the 2.5D visual servoing (Equation 41).

In the shortest path 3D, the position based control is used directly to control the translation of the camera. To control the rotation around x and y axis is used one virtual point which lies in the zero of the object frame; this point helps the control scheme to keep the target in the image. The rotation over the optical axis is controlled using the $^{c}u_{zc*}\,\theta$, as in 2.5D visual servoing, which should be driven to zero.

The coordinates of the virtual point (center of the target plane) in the camera frame are $\left({}^{c}X_{o}, {}^{c}Y_{o}, {}^{c}Z_{o}\right)$ and (x, y) are the image coordinates of the same point

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{{}^{c}Z_{o}} \cdot \begin{pmatrix} {}^{c}X_{o} \\ {}^{c}Y_{o} \end{pmatrix} \qquad \text{……………Equation (56)}$$

And its velocity in the image screen is related with the camera velocity, v , by

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = -\begin{pmatrix} -\frac{1}{{}^{c}Z_{o}} & 0 & \frac{x}{{}^{c}Z_{o}} & x \cdot y & -\left(1+x^{2}\right) & y \\ 0 & -\frac{1}{{}^{c}Z_{o}} & \frac{y}{{}^{c}Z_{o}} & \left(1+y^{2}\right) & -x \cdot y & -x \end{pmatrix} \cdot v \qquad \text{……………Equation (57)}$$

The task vector is $e_{[6x1]} = \left({}^{c}X_{c*}, {}^{c}Y_{c*}, {}^{c}Z_{c*}, x, y, {}^{c}u_{zc*}\theta\right)^{T}$ and the transformation from the initial position to the desired can be obtain, as in PBVS, from the composition of the transformation coordinates from the current position to the target with the desired position to it

$${}^{c}T_{c*} = {}^{c}T_{o}\,{}^{o}T_{c*} = \begin{bmatrix} {}^{c}R_{o} & {}^{c}t_{o} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^{o}R_{c*} & {}^{o}t_{c*} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^{c}R_{c*} & {}^{c}t_{c*} \\ 0 & 1 \end{bmatrix} \qquad \text{……………Equation (58)}$$

The interaction model is given by

$$\dot{e}_{[6x1]} = \left({}^{c}\dot{X}_{c*}, {}^{c}\dot{Y}_{c*}, {}^{c}\dot{Z}_{c*}, \dot{x}, \dot{y}, {}^{c}\dot{u}_{zc*}\theta\right)^{T} = J_{SP} \cdot v \qquad \text{……………Equation (59)}$$

The image Jacobian, J_{SP} is defined as

$$J = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -\frac{1}{Z} & 0 & \frac{x}{Z} & x \cdot y & -\left(1 + x^2\right) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & 1 + y^2 & -x \cdot y & -x \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$ ……………Equation (60)

In which Z is the distance to the virtual point, ${}^{c}Z_{o}$.

Expressing it in the Cartesian coordinates

$$(X, Y, Z)^T = \left({}^{c}X_o, {}^{c}Y_o, {}^{c}Z_o\right)^T$$

$$J = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -\frac{1}{Z} & 0 & \frac{X}{Z^2} & \frac{X \cdot Y}{Z^2} & -\left(1 + \frac{X^2}{Z^2}\right) & \frac{Y}{Z} \\ 0 & -\frac{1}{Z} & \frac{Y}{Z^2} & 1 + \frac{Y^2}{Z^2} & -\frac{X \cdot Y}{Z^2} & -\frac{X}{Z} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$ ……………Equation (61)

Then, the control law is given by

$$v = -\lambda \cdot J_{SP}^{-1} \cdot e$$

Due to the simple mathematical definition that J_{SP} has, it is possible to define directly its inverse, J_{SP}^{-1} which facilitate its implementation

$$J_{SP}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -\frac{X \cdot Y}{Z \cdot P} & \frac{X^2 \cdot Z^2}{Z \cdot P} & -\frac{Y}{P} & -\frac{X \cdot Y}{P} & \frac{X^2 + Z^2}{P} & \frac{X}{Z} \\ -\frac{Y^2 + Z^2}{Z \cdot P} & \frac{X \cdot Y}{Z \cdot P} & \frac{X}{P} & -\frac{Y^2 + Z^2}{P} & \frac{X \cdot Y}{P} & \frac{Y}{Z} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$ ……………Equation (62)

With $P = \frac{X^2 + Y^2 + Z^2}{Z^2}$. ……………Equation (63)

Attending in J_{SP} matrix, its determinant is

$$|Z| = \frac{X^2 + Y^2 + Z^2}{Z^2}$$ ……………Equation (64)

which it is never zero, except when the camera is on the zero of the object frame, which it is not possible. This ensures global convergence of the method.

In the paper where this method was presented, [35], its performance is compared with two other shortest path methods: PBVS and Deguchi visual servoing, [37]. The simulations were done with six points on the target plane [9].

For rotations around the optical axis, both methods had almost the same convergence time and the same feature excursion, i.e. maximum distance of a point from the center of the image. The same conclusions were gotten for a translation along the optical axis [9].

For a rotation around an axis perpendicular to the optical axis in the camera frame, both methods were a similar performance. The execution time was proportional to the rotation and in that case, the camera translation was almost zero [9].

For a rotation around an axis coplanar with the target plane (feature plane rotation), the SP had a little better performance than PBVS in execution time and in maximum features excursion [9].

For a generic motion the SP had also a better performance than PBVS in execution time and maximum features excursion; the difference is bigger with the increase of the initial angle of rotation; in these cases, in PBVS, the points often leave the camera's field of view [9].

All these evaluations were done for a perfect system, without a presence of any perturbation or an unknown parameter [9].

CHAPTER THREE

SYSTEM COMPONENTS

CHAPTER THREE

SYSTEM COMPONENTS

3.1 introduction

The aim of this project is to design a robotic arm visual servoing system using image processing instead of passive sensor (Visual Servoing) which gives better flexibility for the robotic arms applications , such system consist of a robotic arm , an arm motion controller and an image processing controller.

There are many methods to implement a visual servoing controllers which has been discussed in chapter 2 the main two methods are Position based visual servoing and image based visual servoing as shown in figure 3.1 [25]

In position base visual servoing the image processing controller is processing the image from the environment and try to find the suitable position of the object that can be passed to the motion controller as an error signal, in image base visual servoing the image controller try to extract an image feature that the motion controller can use to control the robotic arm.

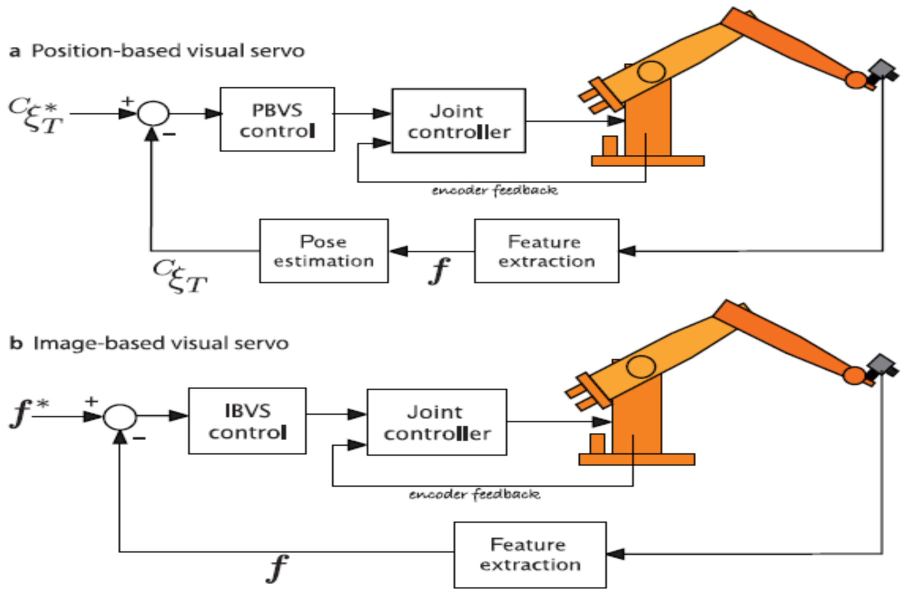


Figure 3.1: The two distinct classes of visual servo system (Photo from: [13])

In this project the position based visual servoing approach has been used due to easiness of building a robotic arm controller based on position error , the project is composed of handmade three joints robotic arm that build out of hard plastic as prove of concept , the arm controlled by TowerPro SG90 - Micro Servo Motor this servo is controlled from arduino board that gives a PWM signal represent the desired angle of the motor , this angles is calculated by a personal computer that runs MathWorks MATLAB Softrware that act as image controller and part of motion controller ,the image for this compute is taken from the environment using an ordinary USB WebCam , the block diagram of the system is shown in figure 3.2 and the full list of system component is showen in table 3.1

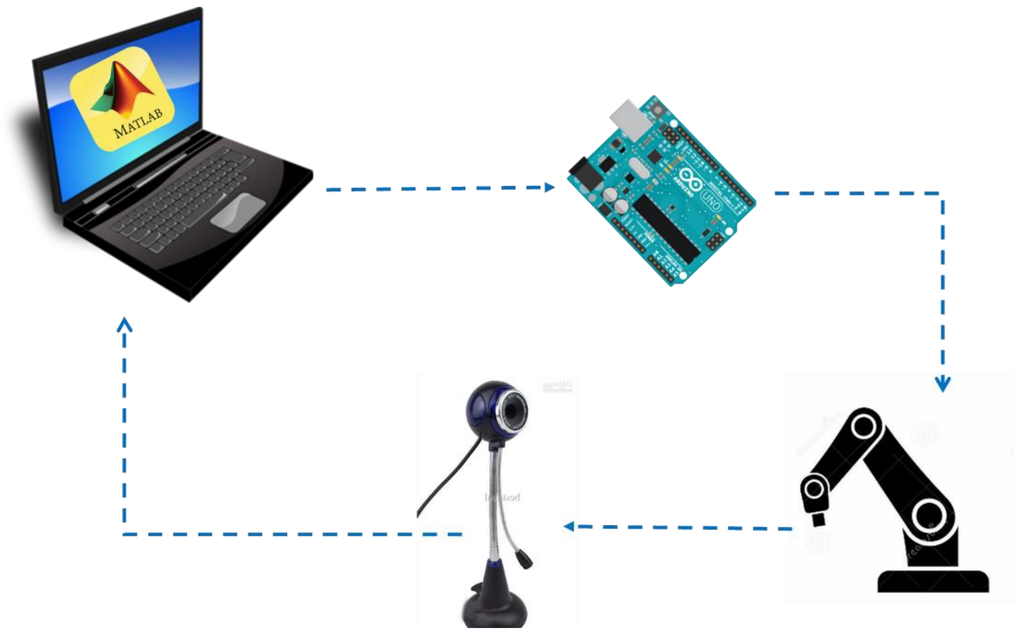


Figure 3.2: The system block diagram

Table 3.1: System component

No.	Component	Qty.	Description
Mechanical Parts			
1	The Base	1	Made of Wood
2	The force bearing	1	
3	Link 1	1	Made of Wood
4	Link 2	2	Made of Hard Plastic
5	Link 3	1	Made of Hard Plastic
Electrical Parts			
1	Joints Servo Motors	3	
2	Arduino Control Board	1	Uno or Nano
3	Web Cam	1	
Software Parts			
1	Arduino Board Software (PDE)		
2	MATLAB Software		
3	Image Acquisition Toolbox		MATLAB Toolbox
4	Image Processing Toolbox		MATLAB Toolbox
5	RobotVision Toolbox		3rd party
6	Arduino IO Library		3rd party

3.2 Robot Construction

The main purpose just to prove the concept of moving the robotic arm (represent the motion) so no big care taken to choose the dimension nor the material , the choose of the joint number and direction desired to be as close as possible to the famous industrial robot PUMA560 and only the first three links which locate the tool (the end effector),the choosing of material was based on the availability in hand in the time of building , the base of the robot cute out of compressed wood and we used a Ball Thrust Bearings on top of wooden base to form the rotating base of the arm (first joint) , the first link is built also from the same wood ,the second and third link are built from hard type of plastic , the dimensions of the robot shown in figure 3.3

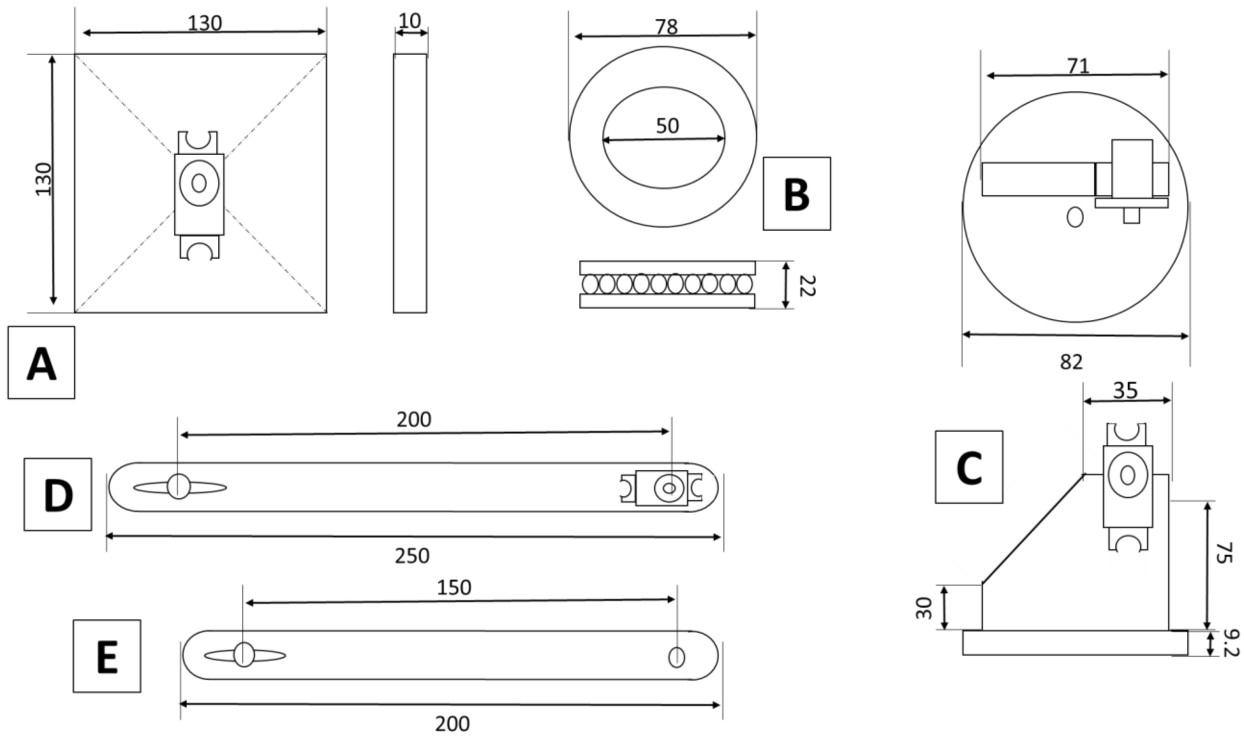


Figure 3.3: Shows the Dimensions of the different parts of the arm (A) The arm base (B) The thrust bearing ; (C) The first link ; (D) The second link ; (E) The third link

3.3 Servo motors

The servo motors was used to rotate the arm joints are Tower Pro SG90 - Micro Servo Motor, it’s a happiest R/C small motor (Figure3.4) The R/C (radio-controlled) “hobby” servo is a prepackaged analog DC servo with potentiometer feedback. It is generally takes less than 10 W of input power, making it quite small, limiting its use to hobby-type applications. However, because of its ease of use and standardized drive specifications, these devices have recently become popular in the realm of education, robotics

Figure 3.4: Tower Pro SG90 - Micro Servo Motor

3.3.1 Anatomy of an RC Servo

Figure 3.5 shows a cross-sectional view of the mechanical parts of an RC servo. The output shaft of a small DC motor is connected through a multiple reduction gear train to the package output shaft. On the external end of the output shaft, a spline is cut, to connect various actuator arms or "horns."

A single-turn potentiometer is connected to the internal end of the output shaft, to provide a position feedback signal to the electronics. Mechanical stops molded into the package prevent the output shaft from turning more than 180°. The input to the device is three-wire, consisting of VCC, ground, and the command signal. [44]

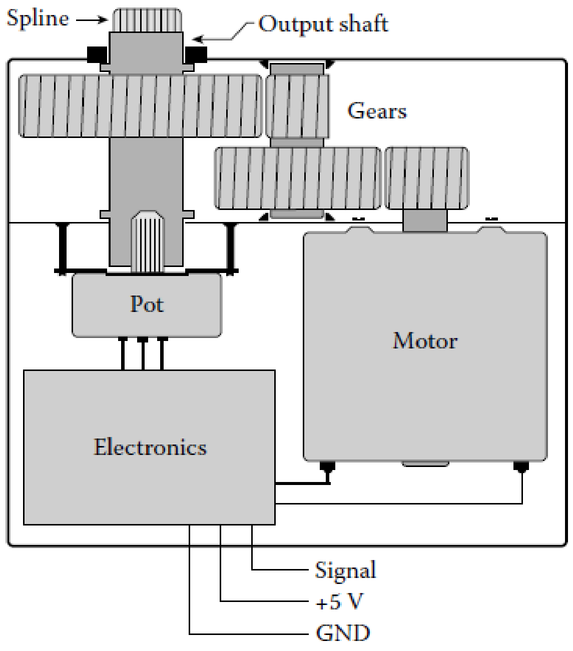


Figure 3.5: Electromechanical schematic for an RC servo (Photo from: [38])

The specification of the servo motor used in this project are

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 µs
- Temperature range: 0 ºC – 55 ºC

The signal wave form used to control the motor position is shown in figure 3.6

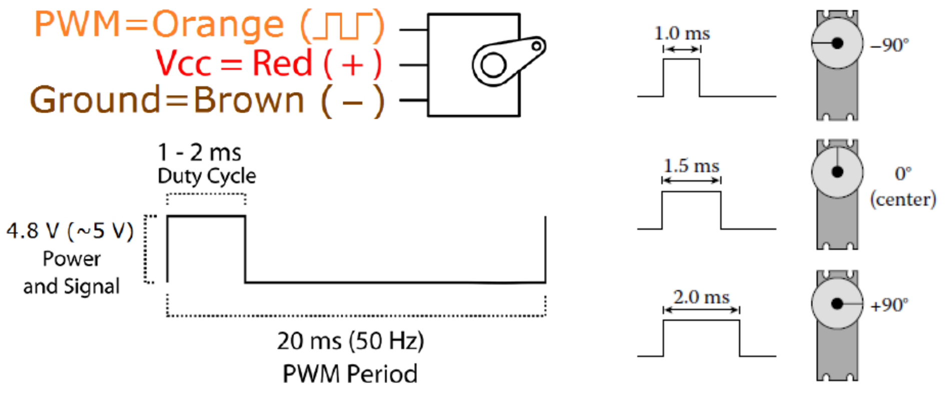


Figure 3.6: The Signal to drive the Servo Motor (Photo from: [38])

3.3.2 How Servo motor works

The simplicity of a servo is among the features that make them so reliable. The heart of a servo is a small direct current (DC) motor, similar to what you might find in an inexpensive toy. These motors run on electricity from a battery and spin at high RPM but normally have very low torque. An arrangement of gears takes the high speed of the motor and slows it down while at the same time increasing the torque. The gear design inside the servo case converts the output to a much slower rotation speed but with more torque. Gears in an inexpensive servo motor are generally made of plastic to keep it lighter and less costly (Plastic Gears can be seen in Figure 3.7).

A positional sensor on the final gear is connected to a small circuit board (can be seen in Figure 6.8). The sensor tells this circuit board how far the servo output shaft has rotated. The electronic input signal from the controller (Arduino in this research) feeds into that circuit board. The electronics on the circuit board decode the signals to determine how far the user wants the servo to rotate. It then compares the desired position to the actual position and decides which direction to rotate the shaft so it gets to the desired position

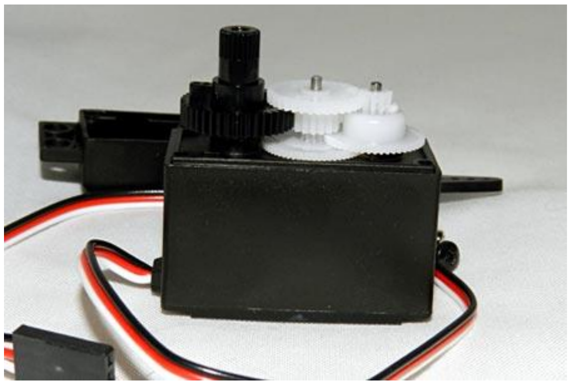


Figure 3.7: The Happiest Servo motor Gears

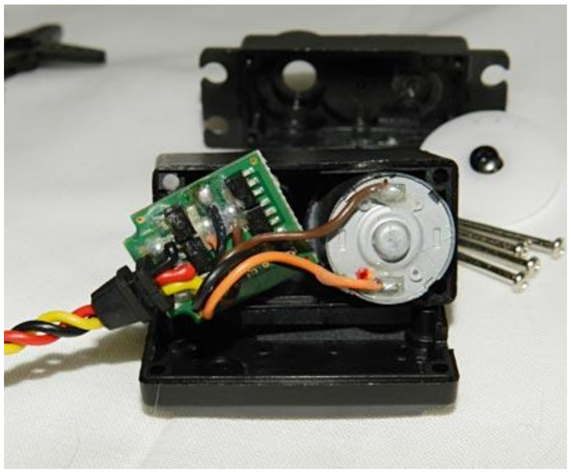


Figure 3.8: The circuit board and DC motor in servo

To demonstrate servo motor control principle consider an example of servomotor that we have given a signal to rotate by an angle of 45° and then stop and wait for further instruction, The shaft of the DC motor is coupled with another shaft called output shaft, with help of gear assembly. This gear assembly is used to step down the high rpm of the motor's shaft to low rpm at output shaft of the servo system (Figure 3.9 (a)).

The voltage adjusting knob of a potentiometer is so arranged with the output shaft by means of another gear assembly, that during rotation of the shaft, the knob also rotates and creates a varying electrical potential according to the principle of potentiometer. This signal i.e. electrical potential is increased with angular movement of potentiometer knob along with the system shaft from 0° to 45°. This electrical potential or voltage is taken to the error detector feedback amplifier along with the input reference commends i.e. input signal voltage (Figure 3.9 (a)).

As the angle of rotation of the shaft increases from 0° to 45° the voltage from potentiometer increases. At 45° this voltage reaches to a value which is equal to the given input command voltage to the system. As at this position of the shaft, there is no difference between the signal voltage coming from the potentiometer and reference input voltage (command signal) to the system, the output voltage of the amplifier becomes zero, As seen in figure 3.9 (c) the output electrical voltage signal of the amplifier, acts as input voltage of the DC motor. Hence the motor will stop rotating after the shaft rotates by 45°. The motor will be at this rest position until another command is given to the system for further movement of the shaft in desired direction.

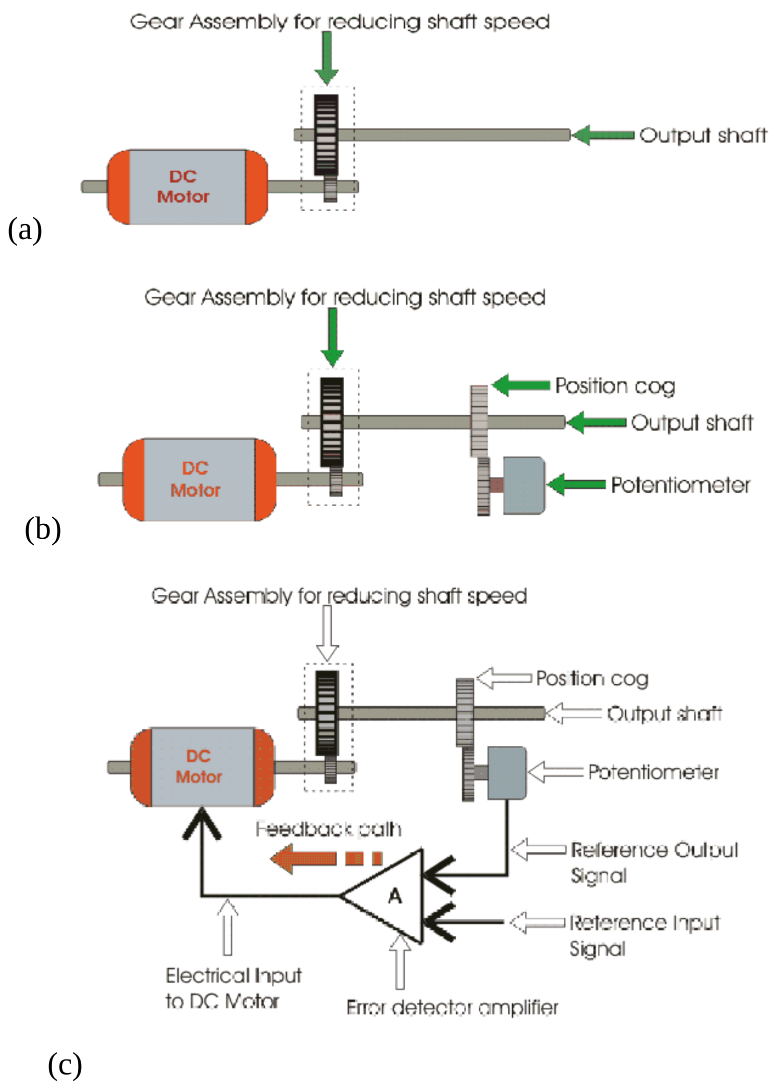


Figure 3.9: Servo motor Demonstration

3.4 Arduino IO Library

ARDUINO IO (Also Known As: "TETHERED" MATLAB SUPPORT PACKAGE FOR ARDUINO): The package allows using an Arduino connected to the computer to perform Analog and Digital Input and Output, (and command motors) from MATLAB.

While earlier version of the package were targeted at smaller boards like the Uno and Nano, since August 2012 (Ver 3.3) the package also works right out of the box on the Mega boards, without requiring any big modification.

3.4.1 ADIOES.PDE

The ADIOES.PDE sketch is the "server" program that continuously run on the microcontroller board. It listens for MATLAB commands arriving from the serial port, executes the commands, and, if needed, returns a result , Arduino IDE can be used to upload the ADIOES.PDE File to the Arduino board .

The following sketches are provided with the package:

- adio.pde : analog and digital IO, plus basic serial commands only
- adioe.pde : adio.pde + encoders support
- adioes.pde : adioe.pde + servo support
- motor_v1.pde : adioes.pde + afmotor v1 shield
- motor_v2.pde : adioes.pde + afmotor v2 shield

In most cases the adioes.pde will cover all you need. The last two sketches are needed for the Adafruit Motor Shield (version 1 and 2 respectively). The first 2 sketches are mostly provided in case your specific platform does not support servos or encoders and as a better (simpler) starting point for people that want to customize a sketch for their own purposes. In the latter case, the "roundtrip" function is probably the easiest place to get started to introduce custom code.

Also the Arduino Package can be installed in MATLAB is by lunching "install_arduino" command in command prompt , considering that MALAB and the user have the required rights to access file paths and serial ports .

3.4.2 Using The Package

This Library can be used over serial port by connecting the Arduino Board to the computer throw a USB cable and installing the serial port drivers .

From MATLAB, command a=arduino('port') where 'port' is the COM port to which the Arduino is connected to, e.g. 'COM5' or 'COM8' can be used to prepare the communication , then the commands a.pinMode, a.digitalRead, a.digitalWrite, a.analogRead, and a.analogWrite, can be used to respectively change the mode (input/output) of each pin, perform digital input, digital output, analog input, and analog output, also commands such as a.servoAttach, a.servoStatus, a.servoWrite, a.servoRead, and a.servoDetach can be used to respectively attach a servo to a PWM pin, get its status (attached/detached) move it to a certain angle, and read its position.

Finally, a.delete can be used to delete the Arduino object, (and free up the serial port) when the session is over.

CHPTER FOUR

SYSTEM IMPLEMENTATION AND RESULTS

CHAPTER FOUR

SYSTEM IMPLEMENTATION AND RESULTS

4.1 introduction

The main aim of the project is to design an image servoing controller to control a robotic arm , in the project a simple 3 degree of freedom robotic arm has built (as discussed in 3.2) just to prove the concept and designed and implement a visual servoing controller using Matlab environment .

4.2 building the robotic arm

the base of the robot cut out of compressed wood and a Ball Thrust Bearings used on top of wooden base to form the rotating bas of the arm (first joint), the first link was built also from the same wood, the second and third link was built from hard type of plastic, figures 4.1 shows the parts of the arm and figure 4.2 shows the assembled arm

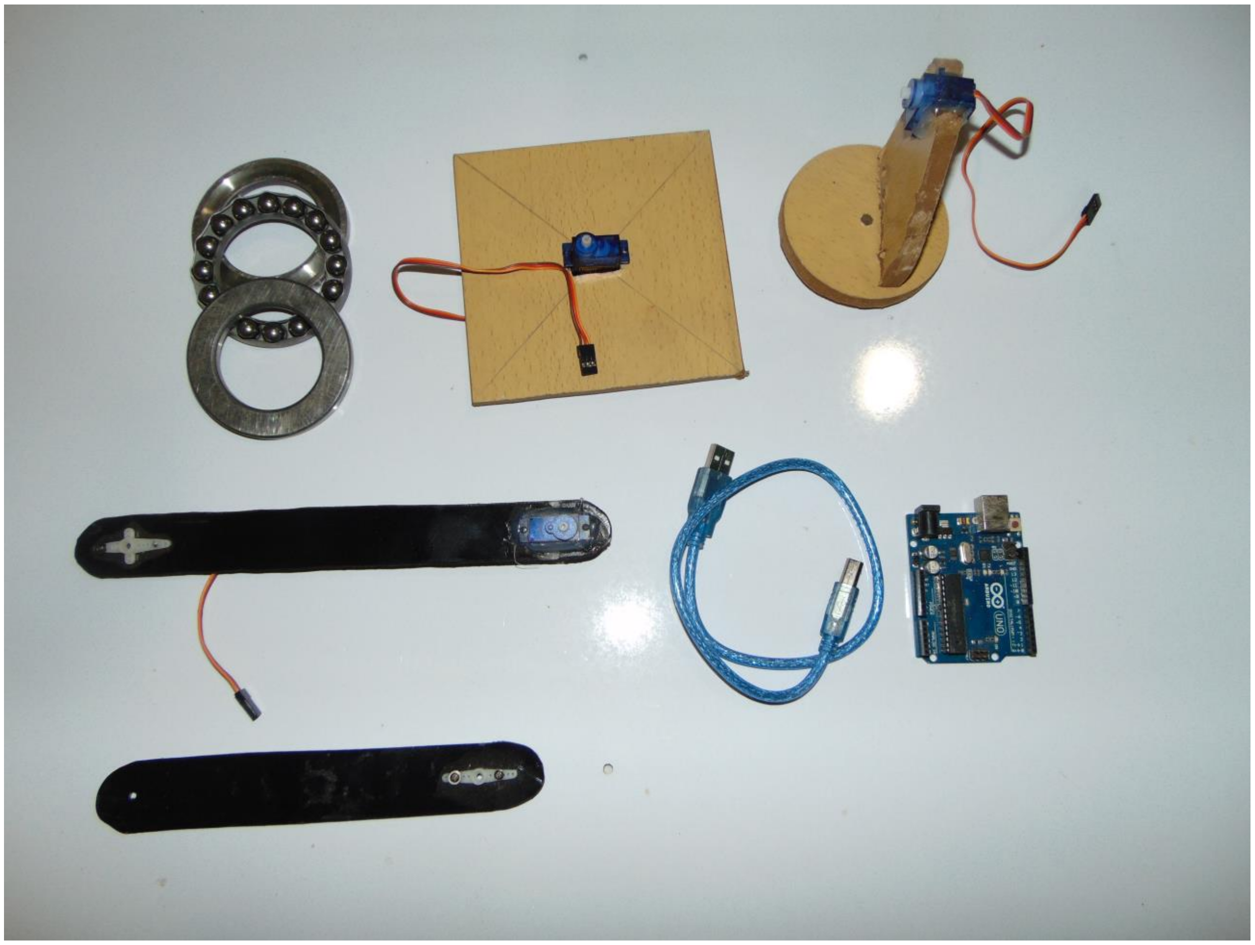


Figure 4.1: The different parts of the arm

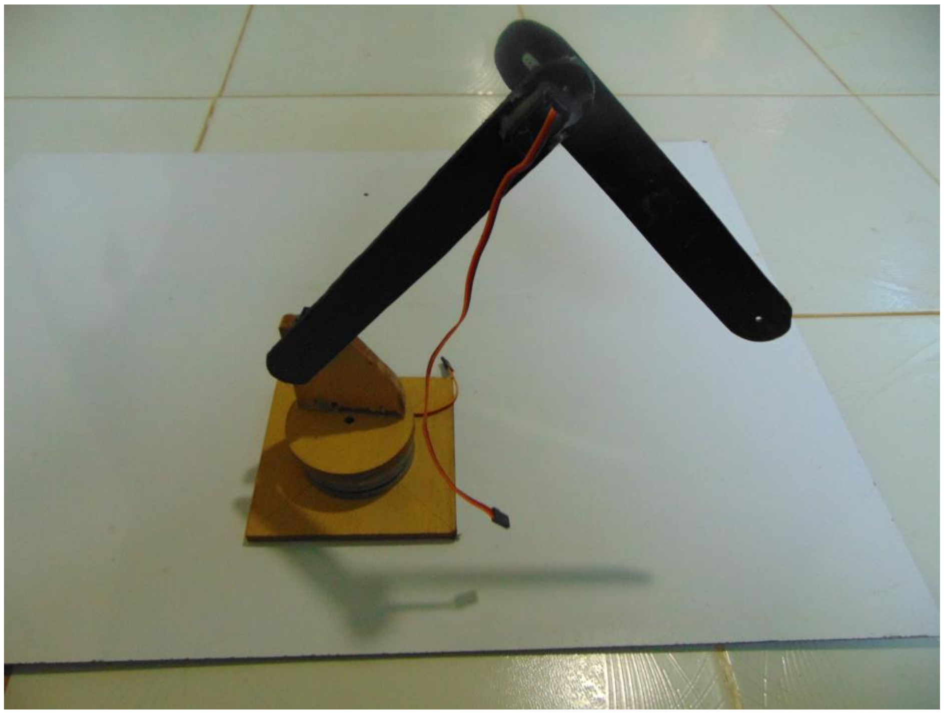


Figure 4.2: The assembled arm

To move the arm three micro servo motor was used and they controlled by a PWM signal generated from an Arduino Uno Board , this board gets the angels of the servo from the computer via a USB cable , figure 4.3 shows the connection diagram

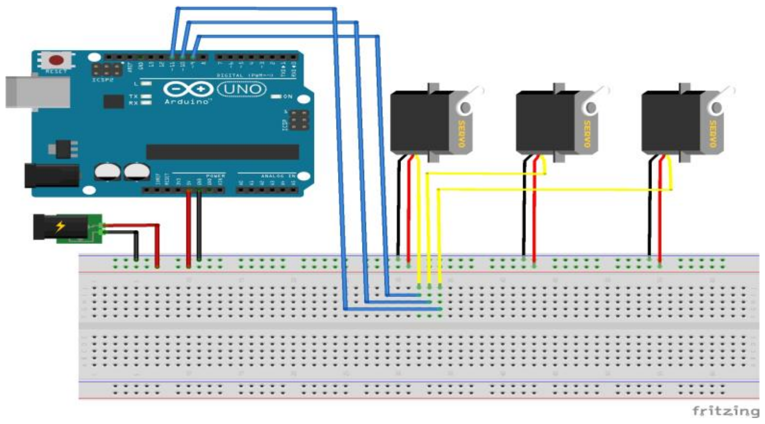


Figure 4.3: The Connection of the servos to the Arduino board

4.3 Calculating the arm parameters

The lengths of the physical arm used in millimeters is shown in figure 4.4 where the joints initial angles for joint 1,2,and 3 respectively is $[0\ ,\ \pi/2\ ,\ 0]$.

The initial position of the arm taken as when all its links pointing upwards as shown in figure 4.5 , the DH parameters of the arm shown in table 4.1 below

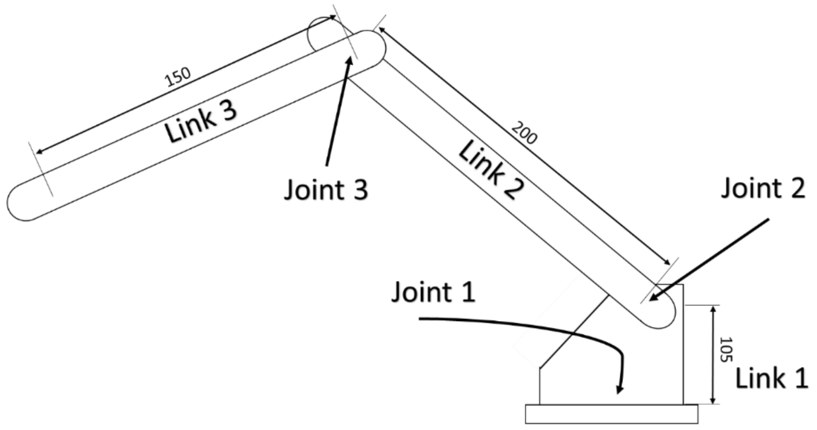


Figure 4.4: The lengths of the physical arm used in millimeters and the joints positions

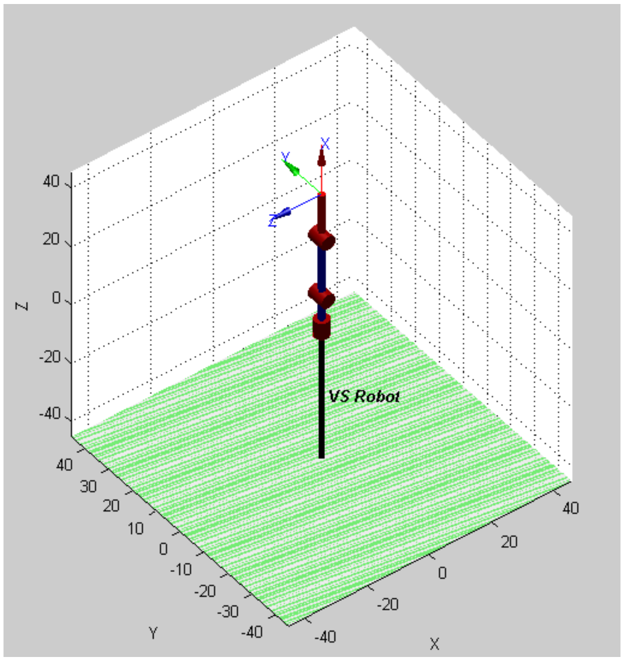


Figure 4.5: The arm initial position

Link j	θ_j	d_i	a_i	α_i	σ_i
1	q_1	10.5	0	$\pi/2$	0
2	q_2	0	20	π	0
3	q_3	0	15	$\pi/2$	0

Table 4.1 Shows the D-H Parameters of the arm

4.4 implementing the image controller

The duty of the image controller is to analysis the images comes from the camera and define the position of the target , analysis the angles needed for the robot to achieve that position and send this angels to the Arduino Board , this has done as next algorithm (figure 4.6)

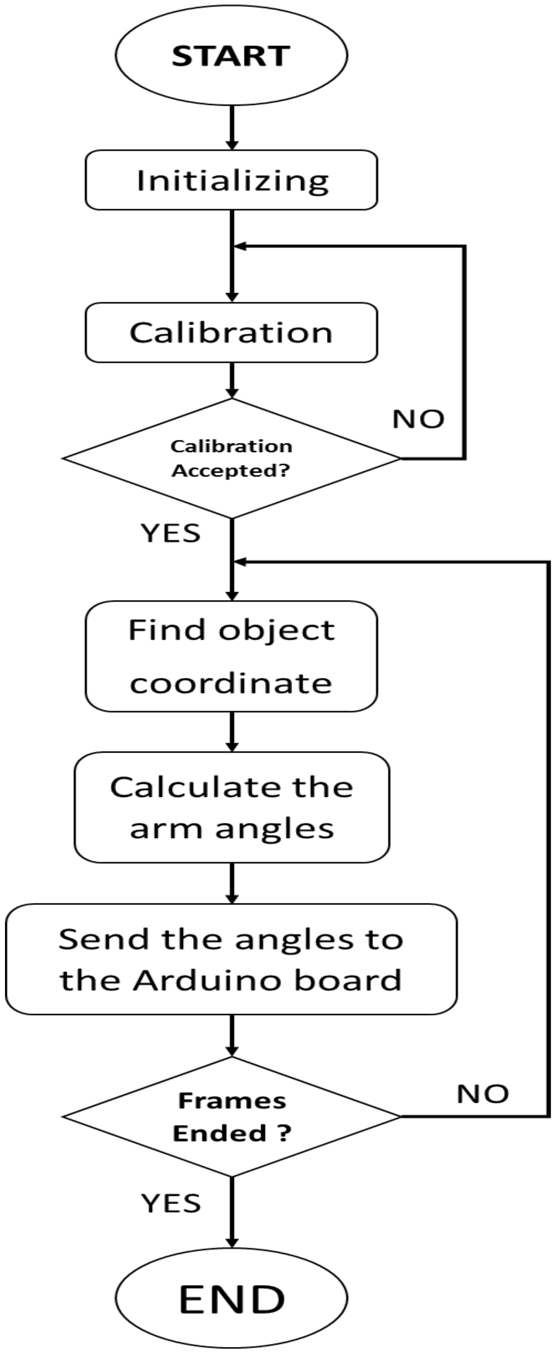


Figure 4.6: The Image controller algorithm

4.4.1 Initialization

This part of the controller program run at first start, this part is responsible of preparing the video object, the robotic arm parameter, the Arduino object and the constant needed during execution of the controller code

4.4.2 Calibration

Because it's very difficult to guarantee the same position and angle of the arm each time, a calibration method have been used, this done by well-known parameter reference placed in front of the arm (figure 4.7), by analyzing this reference first a scaling factor can be added to compensate for the camera position and angle

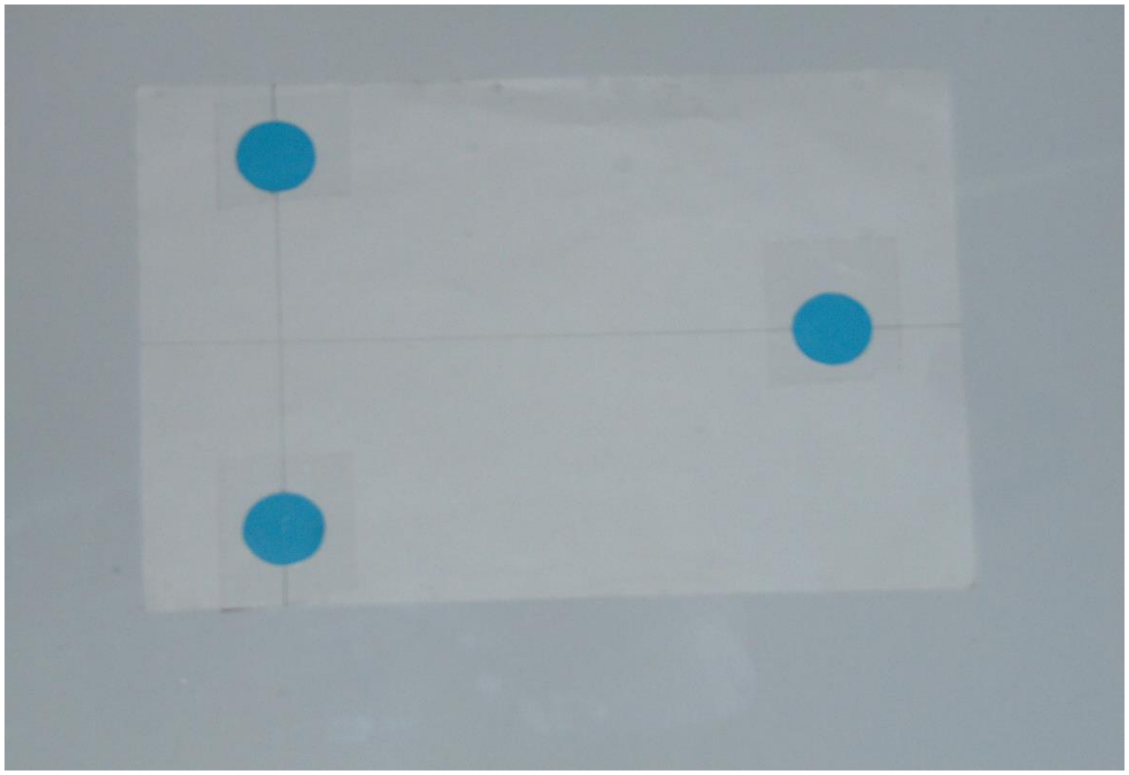


Figure 4.7: The calibration reference

The Dimensions of the calibration reference used is shown in figure 4.8 below, the calibration function return two major information's: the angle of rotation of the image, the mm to pixels scale and the point where the center of the arm .

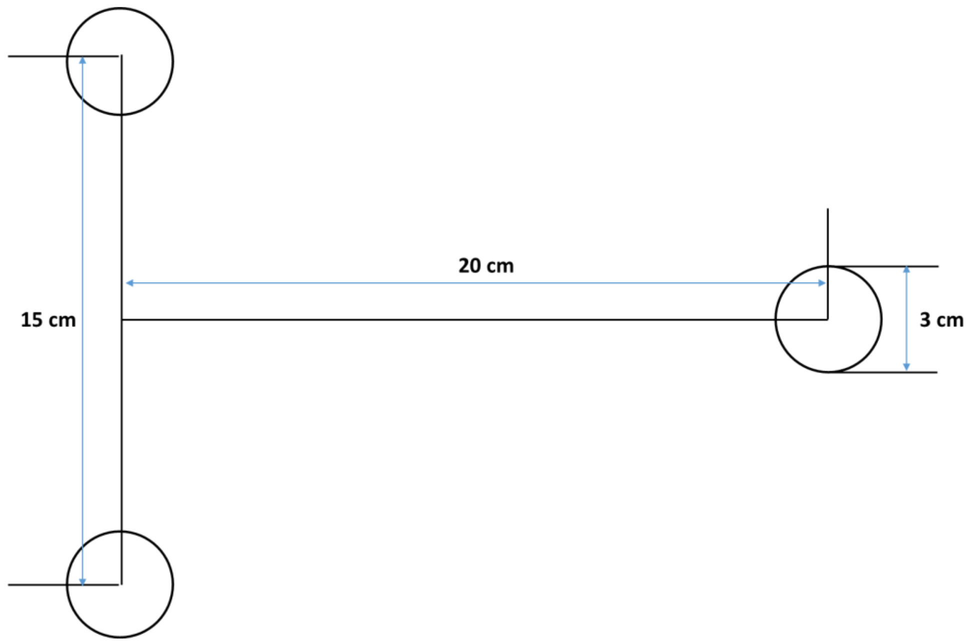


Figure 4.8: The calibration reference dimensions

The calibration process is done by first finding the three blue circles, then calculating the distance between them the direction of the smallest distance is where the arm base , then the center point of this distance is calculated by the equation 4.1 and the distance to the third circle is calculated and used to find mm to pixel scale .

$$Point\ of\ center\ (PC) = \left[\frac{\max(X_{P1}, X_{P2)}\ + \min(X_{P1}, X_{P2)}}{2}\ , \frac{\max(Y_{P1}, Y_{P2)}\ + \min(Y_{P1}, Y_{P2)}}{2}\right] \ldots\ldots..equation\ 4.1$$

The center of the arm base in real world is known to be set on certain distance to the left of the center of the smallest line of the triangle of the calibration reference (defined in the initialization constants), if the PC point and the center of the third circuit are in vertical or horizontal line then constant added to PC to calculate the arm center, else the slop is calculated to calculate the center.

4.4.3 Find the object

The final goal of this part is to define the position of the object relative to the robot center, this done throw three steps: first find the object in video coordinate (which has the zero in top left corner) using image processing (shown in figure 4.9) and then move it to an image coordinate (which has the zero in bottom left corner) and finally move it to the robot coordinate (which has the zero and rotation angle calculated in calibration stage)

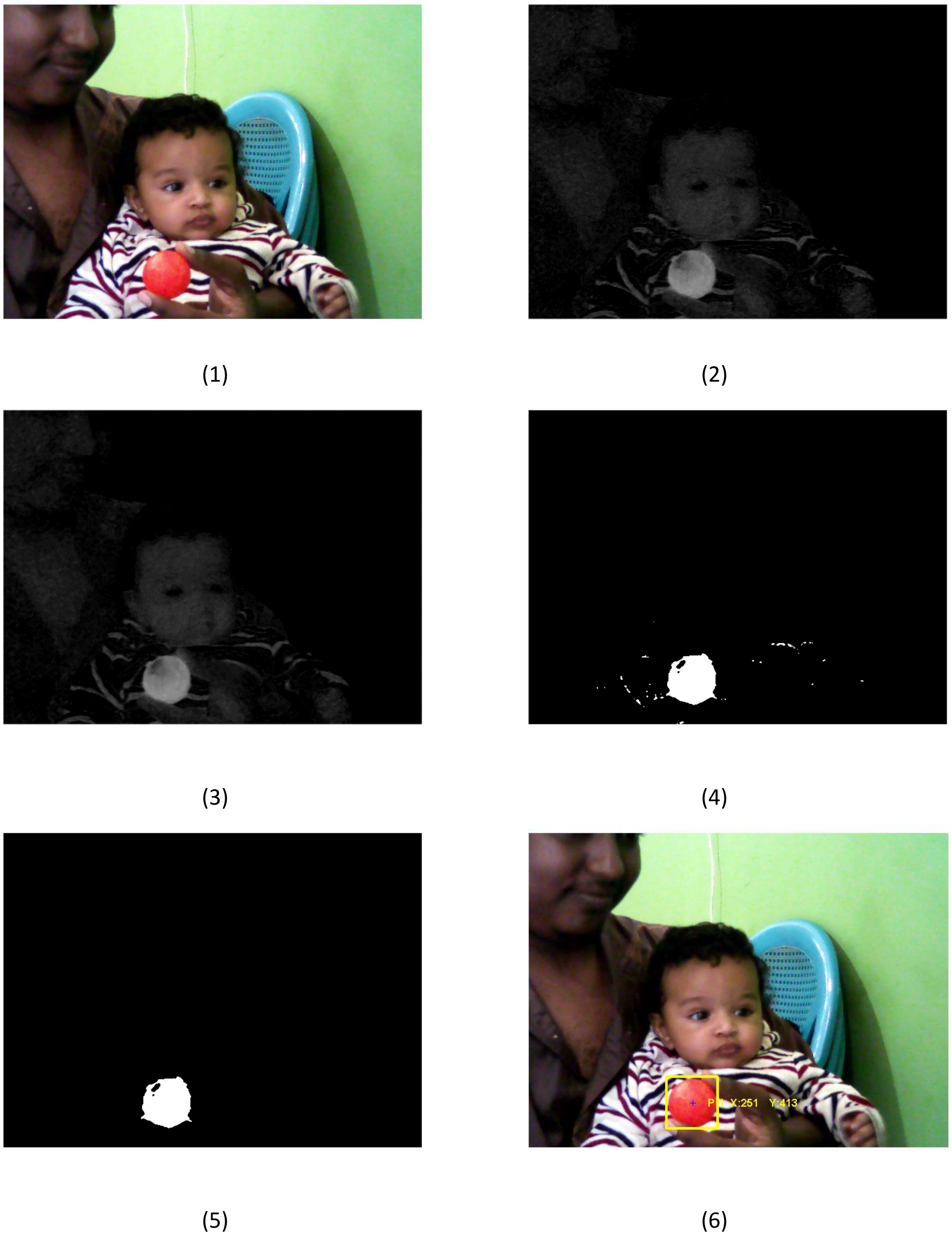


Figure 4.9: The image processing steps used in the image controller: (1) The original image

(2) Subtract the color layer from a gray image ; (3) Appling a median filter

(4) Convert to black and white ; (5) Remove small objects ; (6) Mark founded objects

4.4.4 Calculate the angles of joints

Using the kinematics of the robot and functions from Robot Vision Tool box this part calculate the joint angles needed to achieve the target point and simulate this motion in a graph (figure 4.10)

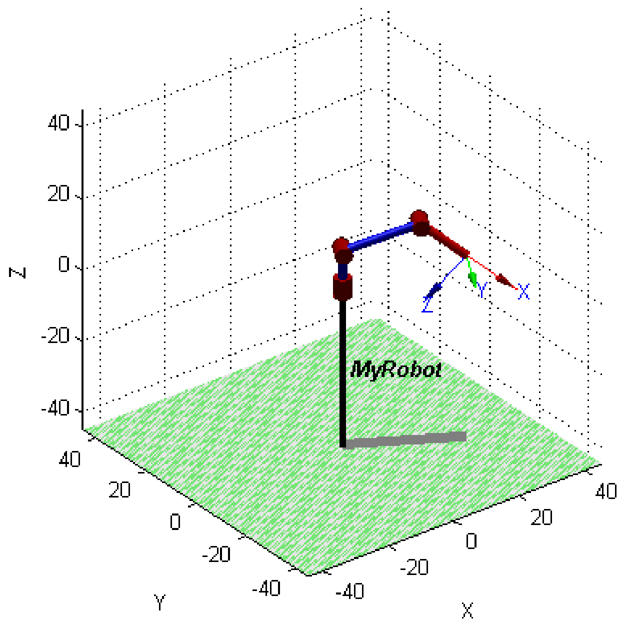


Figure 4.10: The simulated arm by RobotVision Toolbox

4.4.5 Send angles to the Arduino

The final part is to send the calculated angels to the Arduino board to convert it to a PWM signal to drive the servo motor

The three last steps are repeated for certain amount of image frames and then stops, the full Matlab code is shone in appendix A

4.5 Results

As discussed in this chapter the implementation of position based visual servoing is carried out by designing and fabricating a small robotic arm model, wiring a code which use the image captured by the web cam and analyze it to find the position of the desired object, convert the object position into the robot coordinate and using the reverse kinematics calculating the angles needed for the joints to reach that position and send it to the arm controller to execute it.

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion:

Grasping and manipulation of objects is a necessary capability of any robot performing realistic tasks in a natural environment. This thesis presented an application of visual servoing, image based approach was used that benefits from other common sensors to provide information about the positioning of an object.

Traditional visual servo control systems can be divided into two common approaches, image-based (IBVS) and position-based visual servoing (PBVS). As the names suggest, IBVS designs motion based on image-space feedback and PBVS designs motion based on 3D Cartesian-space feedback. In this thesis I used position-based visual servoing (PBVS) approach.

For build this system first an illustrator robotic arm was built out of easy to get materials and developed positioning controller for it.

Secondly an image processing algorithm was developed to detect and track a colored object and then developed a calibration method to detect the position and the direction of the robot arm

Finally an algorithm was developed to track the colored object position, convert it to angels to drive the joints servo motors and send it to the controller board to execute it.

The project took an advantage of MathWorks® MATLAB software and its toolboxes such as image accusation toolbox which makes taking an image from a camera an easy job, and the image processing toolbox which offer a rich and useful function to process a function

And also took an advantage of Peter Corke Robotic toolbox in calculating the inverse kinematics and simulate the position of the arm.

And existence of Arduino and ArduinoIO toolbox makes controlling the servo motors that controls the arm joints an easy to do just by sending the servo angle to it.

5.2 Recommendation:

1) It would be useful increasing the number of cameras to do 3D visual servoing instead of 2D servoing because 3D servoing is more realistic and near to human
2) And also would be useful to use other feature else than color because it’s easy to be disturb

3) It would be useful to use a dedicated hardware like FPGA base hardware which can bring a parallel processing or using a custom ASIC’s specialized in image processing , the dedicated hardware will save a lot of processing time because it’s one task only. Also it would be useful for the university and department to develop a soft-core FPGA based image processor, that will help in researches related to image processing application and save a lot of researchers times.
4) It would be useful and time saving the university and department to develop and build a reference robots that can be used for future robotic researches without losing times in redesign and reanalysis robots.
5) Visual servoing is new and promising field for researchers a lot of things can be done starting from new approaches (rather than IBVS and PBIS) to increase the accuracy and process speed to implement a visual servoing in certain fields like assembly manufacturing, electronic manufacturing, Remote hazardous material handling, mobiles and surveillance and a lot of other fields.

REFERENCES

REFERENCES

[1] J. Hill and T. W. Park, (1979), *“Real Time Control of a Robot with a Mobile Camera”*.

[2] S. Hutchinson, G. Hager and P. I. Corke, (October 1996), *“A tutorial on visual servo control”*. IEEE Trans. on Robotics and Automation, 12(5).

[3] F. Chaumette and S. Hutchinson, (2006), *“Visual Servo Control, Part I: Basic Approaches”*, IEEE Robotics and Automation Magazine.

[4] F. Chaumette and S. Hutchinson, (2007), “*Visual Servo Control, Part II: Advanced Approaches*”, IEEE Robotics and Automation Magazine.

[5] Graziano Chesi and Koichi Hashimoto, (2010), “*Visual Servoing via Advanced Numerical Methods”*, Springer.

[6] Jorge Angeles, (2007), “*Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms”*, Third Edition, Springer.

[7] D. Kragic and M. Vincze, (2010), “*Vision for Robotics*”, first edition, now Publishers Inc.

[8] D. Kragic and H. I. Christensen, (2002), “*Survey on Visual Servoing for Manipulation”*. Technical report, Computational Vision and Active Perception Laboratory.

[9] Ricardo Manuel and Candeias da Silva Santos, (2007), “*Evaluation of Visual Servoing Techniques for Robotic Applications”*, Stockholm: KTH Royal Institute of Technology.

[10] Ying Wang, Daniel Ewert, Rene Vossen, and Sabina Jeschke, (August 2015), *“A Visual Servoing System for Interactive Human-Robot Object Transfer”*, Journal of Automation and Control Engineering Vol. 3, No. 4.

[11] Hessa Al-Junaid, (2015) “*ANN Based Robotic Arm Visual Servoing Nonlinear System*” , Procedia Computer Science 62 , Elsevier .

[12] Mona Gridseth, (Fall 2015), “*Visual Task Specification User Interface for Uncalibrated Visual Servoing*”, University of Alberta.

[13] Peter Corke, (2011), “*Robotics, Vision and Control, Fundamental Algorithms in MATLAB®*”, Springer.

[14] Rafael C. Gonzalez, Richard E. Woods and Steven L. Eddins, (2009), “*Digital Image Processing Using MATLAB”*, 2’nd Edition, Gatesmark Publishing.

[15] Peter Corke, (2015), “*Robotic toolbox for matlab*”, Peter Corke.

[16] S. Ganpaty, (November 1984), “*Camera location determination problem”*. Technical Memorandum 11358-841102-20-TM, AT&T Bell laboratories.

[17] W. C. Chang., (2007), "*Precise positioning of binocular eye-to-hand robotic manipulators*", Journal of Intelligent Robot System, 49(1).

[18] E. Malis, F. Chaumette, and S. Boudet, (1999) "*2-21/2-D Visual servoing*", IEEE Trans. on Robotics and Automation, 15(2).

[19] W. F. Xie, Z. Li, X. W. Tu, and C. Perron, (2009) "*Switching control of image based visual servoing with laser pointer in robotic manufacturing systems"*, IEEE Trans. on Industrial Electronics, 56(2).

[20] E. J. González-Gávaln, S. R. Cruz-Ramírez, M. J. Seelinger, and J. J. Cervantes-Sánchez, (2003), *"An efficient multi-camera, multi-target scheme for the three-dimensional control of robots using uncalibrated vision"*. Robotics and Computer-Integrated Manufacturing, 19(5).

[21] D. I. Kosmopoulos, (2011), "*Robust Jacobian matrix estimation for image-based visual servoing"*, Robotics and Computer-Integrated Manufacturing, 27(1).

[22] L. Deng, (2003), "*Comparison of image-based and position-based robot visual servoing methods and improvements*", A PhD dissertation of Electrical and Computer Engineering of University of Waterloo.

[23] P. R. Giordano, A. de Luca, and G. Oriolo, (May 2008), "*3D Structure Identification from Image Moments*", IEEE International Conference on Robotics and Automation, Pasadena, CA, USA.

[24] O. Tahri and F. Chaumette, (December 2005), "*Point-based and region image moments for visual servoing of planar objects"*. IEEE Trans. on Robotics, 21(6).

[25] R. J. Prokop and A. P. Reeves, (September 1992), "*A survey of moment-based techniques for unoccluded object representation and recognition*", CVGIP: Graphical Models and Image Processing, 54(5).

[26] F. Chaumette, (August 2004), "*Image moments: a general and useful set of features for visual servoing*". IEEE Trans. on Robotics, 20(4).

[27] Feddema, J.T., Lee, C.S.G. and Mitchell, O.R., (February 1991), *"Weighted selection of image features for resolved rate visual feedback control"*, IEEE Trans. Robot. Automat., vol. 7.

[28] Kragic, D., (2011), *"Visual Servoing for Manipulation: Robustness and Integration"*, PhD thesis, CVAP, NADA, Royal Institute of Technology, KTH.

[29] Vincze, M., Ayromlou, M. and Kubinger, W., (1999), *"Improving the robustness of image-based tracking to control 3D robot motions"*, in Proceedings of the international Conference on Image Analysis and Processing.

[30] Bray A. J., (1990), *"Tracking objects using image disparities"*, Image and Vision Computing.

[31] Schick, J. and Dickmanns, E.D., (1991*), "Simultaneous estimation of 3D shape and motion of objects by computer vision"*, in Proceedings of the IEEE Workshop on Visual Motion.

[32] Tarabanis, K.A., Allen, P.K. and Tsai, R.Y., (1995), *"A survey of sensor planning in computer vision"*, in IEEE Transactions on Robotics and Automation 11(1).

[33] Vincze, M. and Hager, G, (2000), "*Robust vision for vision-based control of motion*", Wash.: SPIE Optical Engineering Press.

[34] Corke, P. I. and Hutchinson S. A., (August 2001), *"A new partitioned approach to image-based visual servo control"*, IEEE Transactions on Robotics and Automation, vol. 17, no. 4.

[35] Kyrki V., Kragic D., and Christensen H., (October 2004), *"New shortest-path approaches to visual servoing"*, in IEEE Int. Conf. on Intelligent Robots and Systems, Sendai, Japan.

[36] F. Chaumette, (1998), "*Potential problems of stability and convergence in image-based and position-based visual servoing*", In The Conference of Vision and Control, Series Lecture Notes in Control and Information Science, vol. 237, Verlag, New York.

[37] Miller T., (August 1989), *"Real-Time application of neural networks for sensor-based control of robots with vision"*, in IEEE Transactions on systems, man, and cybernetics, vol. 19, no. 4.

[38] Stephen M. Tobin, (2011), *"DC Servos: Application And Design with MATLAB®"*, CRC Press.

APPENDIXES

APPENDIX A

MATLAB CODE

A.1 The final code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%     initialization     %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Clear and initialize the work Space
clear all
clc
v=ver;
toolbox_exist=any(strcmp(cellstr(char(v.Name)), 'Robotics Toolbox'));
if toolbox_exist==0
   path='C:\Program Files\MATLAB\R2013a\toolbox\rvctools';
   addpath(path);
   startup_rvc
end
%Image processing coeficions
Camera_Num=2;
B_th=0.17;
R_th=0.17;
R=1;G=2;B=3;
%Robot Arm coeficions
Link_1=105;
Link_2=200;
Link_3=150;
Rob_Des=200;
Robot_Zero =[0 pi/2 0];
%Arduino coeficions
ComPort='COM17';
Servo_A = 9;
Servo_B = 10;
Servo_C = 11;
%initialaizing Figures
f1=figure('Position',[54 84 600 600],'Name','Robot Model');
f2=figure('Position',[702 84 600 600],'Name','Image Processing');
%initialaizing Camera and its object
vid = videoinput('winvideo', Camera_Num, 'YUY2_640x480');
set(vid, 'FramesPerTrigger', Inf);
set(vid, 'ReturnedColorspace', 'rgb')
vid.FrameGrabInterval = 20;
start(vid);
%initialaizing Robot and its object
L(1)= Link([0 Link_1/10 0 pi/2]);
L(2)=Link([0 0 Link_2/10  pi]);
L(3)=Link([0 0 Link_3/10 pi/2]);
rrr_robot =SerialLink(L , 'name', 'VS Robot');
figure(f1)
rrr_robot.plot(Robot_Zero)
%initialaizing Arduino and its object
Arduino=arduino(ComPort);
Arduino.servoAttach(Servo_A);
Arduino.servoAttach(Servo_B);
Arduino.servoAttach(Servo_C);
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%      Calibration       %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cal=0;
while (cal==0)
    frame=1;
    while(frame<=3)
        original_img = getsnapshot(vid);
        objects=plops(original_img,B,B_th);
        figure(f2)
        imshow(original_img)
        hold on
        for i = 1:length(objects)
            bb = objects(i).BoundingBox;
            bc = objects(i).Centroid;
            rectangle('Position',bb,'EdgeColor','m','LineWidth',2)
            plot(bc(1),bc(2), 'b+')
        end
        hold off

        if length(objects) == 3
            P1=objects(1).Centroid;
            P2=objects(2).Centroid;
            P3=objects(3).Centroid;

            RA=Robot_World_Parameter(P1,P2,P3,Rob_Des);
            o_rotation_Angle=round(RA);

            R_img= imrotate(original_img,o_rotation_Angle);

            R_objects=plops(R_img,B,B_th);
            P1=R_objects(1).Centroid;
            P2=R_objects(2).Centroid;
            P3=R_objects(3).Centroid;

            BB1=R_objects(1).BoundingBox;
            BB2=R_objects(2).BoundingBox;
            BB3=R_objects(3).BoundingBox;

            [RA,Robot_Center,Scale]=...
                Robot_World_Parameter(P1,P2,P3,100);
            %to hear we have rotation angle and rotated image and robot value from
            %rotated image
            figure(f2)
            imshow(R_img)
            hold on
                rectangle('Position',BB1,'EdgeColor','r','LineWidth',2)
                plot(P1(1),P1(2), 'b+')
                rectangle('Position',BB2,'EdgeColor','r','LineWidth',2)
                plot(P2(1),P2(2), 'b+')
                rectangle('Position',BB3,'EdgeColor','r','LineWidth',2)
                plot(P3(1),P3(2), 'b+')
                plot(Robot_Center(1),Robot_Center(2), 'y+')
            hold off
        end
        frame=frame+1;
    end
    in=input('Calibration ok? (y/n)[n]  :  ','s');
    if in=='y'
        cal=1;
    else
        cal=0;
    end
end
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%         traking        %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

frame=1;
while(frame<=30)
    original_img = getsnapshot(vid);
    R_img= imrotate(original_img,o_rotation_Angle);

    object=plops(R_img,R,R_th);
    if length(object)>=1
        R_ball_center=object(1).Centroid;
        R_ball_Xmm = (Robot_Center(1)- R_ball_center(1)) * Scale;
        R_ball_Ymm = (R_ball_center(2)- Robot_Center(2)) * Scale;

        object=plops(original_img,R,R_th);
        R_ball_center=object(1).Centroid;
        R_ball_bound=object(1).BoundingBox;

        figure(f2)
        imshow(original_img)
        hold on
        rectangle('Position',R_ball_bound,'EdgeColor','m','LineWidth',2)
        plot(R_ball_center(1),R_ball_center(2), 'b+')
        tx=text(R_ball_center(1)+15,R_ball_center(2), strcat('  P: ', '  X: ',...
        num2str(round(R_ball_Xmm)), '    Y: ', num2str(round(R_ball_Ymm))));
        set(tx, 'FontName', 'Arial', 'FontSize', 10, 'Color', 'red');
        tx=text(7,10,strcat('Angle : ',num2str(o_rotation_Angle)));
        set(tx, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color', 'yellow');
        tx=text(7,30,strcat('Frame Number : ',num2str(frame)));
        set(tx, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color', 'yellow');

        %finding the inverse kinamatics
        T = Geo_invers_kin(Link_1,Link_2,Link_3,R_ball_Xmm,R_ball_Ymm,40);
        if isreal(T)

            TS(1)=T(1);
            TS(2)=(90-T(2))*-1;
            TS(3)=180-T(3);

            TA(1)=90-T(1);
            TA(2)=T(2);
            TA(3)=180-T(3);

            %Draw info
            tx=text(7,50,strcat('Joints : ',num2str(T(1)) , ' , ',num2str(T(2)) , ' ,
',num2str(T(3))));
            set(tx, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color',
'yellow');
            hold off
            %simulate
            figure(f1)
            rrr_robot.plot(deg2rad(TS));
            %Saturate the servo
            for i=1:length(TA)
                if TA(i)>180
                    TA(i)=180;
                elseif TA(i)<0
                    TA(i)=0;
                end
            end
            %Send to Arduino
            Arduino.servoWrite(Servo_A,TA(1))
            Arduino.servoWrite(Servo_B,TA(2))
            Arduino.servoWrite(Servo_C,TA(3))
        else
            tx=text(7,60,'Object out of Robot Reach');
            set(tx, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 18, 'Color', 'red');
            hold off
        end
    else
        figure(f2)
        hold on
        tx=text(200,200,'There Are No Object');
        set(tx, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 32, 'Color', 'yellow');
        hold off
    end

    frame=frame+1;
end
```

```
%%
stop(vid);
% Flush all the image data stored in the memory buffer.
flushdata(vid);
```

A.2 Function “polps”

This function returns information about color objects in an image

```
function object_info = plops(image,color,threashold)
    diff_im = imsubtract(image(:,:,color), rgb2gray(image));
    diff_im = medfilt2(diff_im, [3 3]);
    diff_im = im2bw(diff_im,threashold);
    diff_im = bwareaopen(diff_im,150);
    bw = bwlabel(diff_im, 8);
    object_info = regionprops(bw, 'BoundingBox', 'Centroid');
end
```

A.3 Function “Robot_World_Parameter”

This function calculate the orientation , the slop and the ropot center depending on the calibration plate pointes

```
function [rotation_Angle,Robot_Center,Scale]=Robot_World_Parameter(P1,P2,P3,mm_D)
tolarence=10;
D1=round(pdist([P1;P2]));
D2=round(pdist([P2;P3]));
D3=round(pdist([P3;P1]));
small=min([D1;D2;D3]);
%Scale=small/150;  %how many mm in pixcel   mm = scale * Pixels    ; Pixels= mm/scale


if small==D1

    PC=[(max([P1(1);P2(1)])+(min([P1(1);P2(1)])))/2 (max([P1(2);P2(2)])+(min([P1(2);P2(2)])))/2];
    LD=round(pdist([P3;PC]));
    Scale=LD/200;
    Pixel_D = mm_D / Scale ;
    if P1(2)>(P2(2)-tolarence) && P1(2)<(P2(2)+tolarence)         %Horezental
        if P3(2)> P1(2)
            %type='HU';
            theta=0;
            Robot_Center(1)=PC(1);
            Robot_Center(2)=PC(2)-Pixel_D;
        else
            %type='HD';
            theta=180;
            Robot_Center(1)=PC(1);
            Robot_Center(2)=PC(2)+Pixel_D;
        end

    elseif P1(1)>(P2(1)-tolarence) && P1(1)<(P2(1)+tolarence)     %Vertical
        if P3(1)> P1(1)
            %type='VR';
            theta=-90;
            Robot_Center(1)=PC(1)-Pixel_D;
            Robot_Center(2)=PC(2);
        else
            %type='VL';
            theta=90;
            Robot_Center(1)=PC(1)+Pixel_D;
            Robot_Center(2)=PC(2);
        end
    else                                              % in angle
        slp=(PC(2)-P3(2))/(PC(1)-P3(1));
        slp=-1*(1/slp);
        slop_angle=atand(slp);
        if slop_angle > 0
           if PC(2)>P3(2)
               %type='Q4';
               theta=-180+slop_angle;
               B=slop_angle;
               Robot_Center(1)=PC(1)-(Pixel_D * sind(B));
               Robot_Center(2)=PC(2)+(Pixel_D * cosd(B));
           else
               %type ='Q2';
               theta=slop_angle;
               B=theta;
               Robot_Center(1)=PC(1)+(Pixel_D * sind(B));
               Robot_Center(2)=PC(2)-(Pixel_D * cosd(B));
           end
        else
            if PC(2)>P3(2)
               %type='Q3';
               theta=180+slop_angle;
               B=90+slop_angle;
               Robot_Center(1)=PC(1)+(Pixel_D * cosd(B));
               Robot_Center(2)=PC(2)+(Pixel_D * sind(B));

           else
               %type ='Q1';
               theta=slop_angle;
               B=90+theta;
               Robot_Center(1)=PC(1)-(Pixel_D * cosd(B));
               Robot_Center(2)=PC(2)-(Pixel_D * sind(B));
           end
        end
    end
%===============================================================================================
elseif small==D2


    PC=[(max([P2(1);P3(1)])+(min([P2(1);P3(1)])))/2 (max([P2(2);P3(2)])+(min([P2(2);P3(2)])))/2];
    LD=round(pdist([P1;PC]));
    Scale=LD/200;
    Pixel_D = mm_D / Scale ;
    if P2(2)>(P3(2)-tolarence) && P2(2)<(P3(2)+tolarence)         %Horezental
        if P1(2)> P2(2)
            %type='HU';
            theta=0;
            Robot_Center(1)=PC(1);
            Robot_Center(2)=PC(2)-Pixel_D;
        else
            %type='HD';
            theta=180;
            Robot_Center(1)=PC(1);
            Robot_Center(2)=PC(2)+Pixel_D;
        end

    elseif P2(1)>(P3(1)-tolarence) && P2(1)<(P3(1)+tolarence)     %Vertical
        if P1(1)> P2(1)
            %type='VR';
            theta=-90;
            Robot_Center(1)=PC(1)-Pixel_D;
            Robot_Center(2)=PC(2);
        else
            %type='VL';
            theta=90;
            Robot_Center(1)=PC(1)+Pixel_D;
            Robot_Center(2)=PC(2);
        end
    else                                                 % in angle
        slp=(PC(2)-P1(2))/(PC(1)-P1(1));
        slp=-1*(1/slp);
        slop_angle=atand(slp);
        if slop_angle > 0
           if PC(2)>P1(2)
               %type='Q4';
               theta=-180+slop_angle;
               B=slop_angle;
               Robot_Center(1)=PC(1)-(Pixel_D * sind(B));
               Robot_Center(2)=PC(2)+(Pixel_D * cosd(B));
           else
               %type ='Q2';
               theta=slop_angle;
               B=theta;
               Robot_Center(1)=PC(1)+(Pixel_D * sind(B));
               Robot_Center(2)=PC(2)-(Pixel_D * cosd(B));
           end
        else
            if PC(2)>P1(2)
               %type='Q3';
               theta=180+slop_angle;
               B=90+slop_angle;
               Robot_Center(1)=PC(1)+(Pixel_D * cosd(B));
               Robot_Center(2)=PC(2)+(Pixel_D * sind(B));

           else
               %type ='Q1';
               theta=slop_angle;
               B=90+theta;
               Robot_Center(1)=PC(1)-(Pixel_D * cosd(B));
               Robot_Center(2)=PC(2)-(Pixel_D * sind(B));
           end
        end
    end

%===============================================================================================
elseif small==D3

    PC=[(max([P1(1);P3(1)])+(min([P1(1);P3(1)])))/2 (max([P1(2);P3(2)])+(min([P1(2);P3(2)])))/2];
    LD=round(pdist([P2;PC]));
    Scale=LD/200;
    Pixel_D = mm_D / Scale ;
    if P1(2)>(P3(2)-tolarence) && P1(2)<(P3(2)+tolarence)         %Horezental
        if P2(2)> P1(2)
            %type='HU';
            theta=0;
            Robot_Center(1)=PC(1);
            Robot_Center(2)=PC(2)-Pixel_D;
        else
            %type='HD';
            theta=180;
            Robot_Center(1)=PC(1);
            Robot_Center(2)=PC(2)+Pixel_D;
        end

    elseif P1(1)>(P3(1)-tolarence) && P1(1)<(P3(1)+tolarence)     %Vertical
        if P2(1)> P1(1)
            %type='VR';
            theta=-90;
            Robot_Center(1)=PC(1)-Pixel_D;
            Robot_Center(2)=PC(2);
        else
            %type='VL';
            theta=90;
            Robot_Center(1)=PC(1)+Pixel_D;
            Robot_Center(2)=PC(2);
        end

    else                                                 % in angle
```

```
        slp=(PC(2)-P2(2))/(PC(1)-P2(1));
        slp=-1*(1/slp);
        slop_angle=atand(slp);
        if slop_angle > 0
           if PC(2)>P2(2)
               %type='Q4';
               theta=-180+slop_angle;
               B=slop_angle;
               Robot_Center(1)=PC(1)-(Pixel_D * sind(B));
               Robot_Center(2)=PC(2)+(Pixel_D * cosd(B));
           else
               %type ='Q2';
               theta=slop_angle;
               B=theta;
               Robot_Center(1)=PC(1)+(Pixel_D * sind(B));
               Robot_Center(2)=PC(2)-(Pixel_D * cosd(B));
           end
        else
            if PC(2)>P2(2)
               %type='Q3';
               theta=180+slop_angle;
               B=90-slop_angle;
               Robot_Center(1)=PC(1)+(Pixel_D * cosd(B));
               Robot_Center(2)=PC(2)+(Pixel_D * sind(B));

           else
               %type ='Q1';
               theta=slop_angle;
               B=90+theta;
               Robot_Center(1)=PC(1)-(Pixel_D * cosd(B));
               Robot_Center(2)=PC(2)-(Pixel_D * sind(B));
           end
        end
    end
end

%rotation_matrix=[cosd(theta) -sind(theta); sind(theta) cosd(theta)];
rotation_Angle=theta;
end
```

A.3 Function “Robot_World_Parameter”

This function calculate the inverse kinematics of the robot by a geography approach to alter a point x,y,z considering the robot center is 0,0,0

```
function T = Geo_invers_kin(L1,L2,L3,X,Y,Z)
A=L1-Z;
P=sqrt(X^2+Y^2);
D=sqrt(P^2+A^2);

B= acosd(((D^2+L2^2)-L3^2)/(2*L2*D));

F1=asind(A/D);
F2=B-F1;

T(1)=90-acosd(X/P);
T(2)=F2+90;
T(3)= acosd(((L3^2+L2^2)-D^2)/(2*L2*L3));

T(1)=round(T(1));
T(2)=round(T(2));
T(3)=round(T(3));
end
```

APPENDIX B
ARDUINU

B.1 What is Arduino?

(From Wikipedia https://en.wikipedia.org/wiki/Arduino Accessed: 12/12/2015)

Arduino is an open-source computer hardware and software company, project and user community that designs and manufactures microcontroller-based kits for building digital devices and interactive objects that can sense and control objects in the physical world.

The project is based on microcontroller board designs, manufactured by several vendors, using various microcontrollers. These systems provide sets of digital and analog I/O pins that can be interfaced to various expansion boards ("shields") and other circuits. The boards feature serial communications interfaces, including USB on some models, for loading programs from personal computers. For programming the microcontrollers, the Arduino project provides an integrated development environment (IDE) based on the Processing project, which includes support for the C and C++ programming languages.

The first Arduino was introduced in 2005, aiming to provide an inexpensive and easy way for novices and professionals to create devices that interact with their environment using sensors and actuators. Common examples of such devices intended for beginner hobbyists include simple robots, thermostats, and motion detectors.

Arduino started in 2005 as a project for students at the Interaction Design Institute Ivrea in Ivrea, Italy. At that time program students used a "BASIC Stamp" at a cost of $100, considered expensive for students. Massimo Banzi, one of the founders, taught at Ivrea. The name "Arduino" comes from a bar in Ivrea, where some of the founders of the project used to meet. The bar, in turn, has been named after Arduin of Ivrea, who was the margrave of Ivrea and King of Italy from 1002 to 1014.

Colombian student Hernando Barragán created the Wiring development platform which served as the basis for Arduino. Following the completion of the Wiring platform, its lighter, less expensive versions were created and made available to the open-source community; associated researchers, including David Cuartielles, promoted the idea. The Arduino's initial core team consisted of Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis.

Arduino boards are available commercially in preassembled form, or as do-it-yourself kits. The hardware design specifications are openly available, allowing the Arduino boards to be manufactured by anyone. Adafruit Industries estimated in mid-2011 that over 300,000 official Arduinos had been commercially produced, and in 2013 that 700,000 official boards were in users' hands

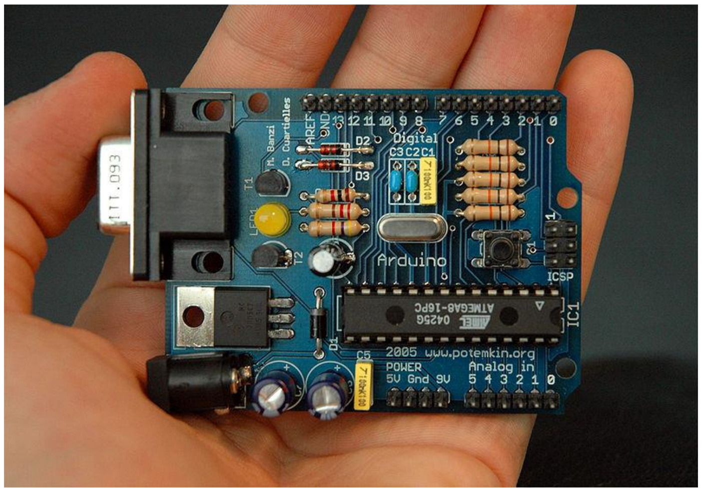


Figure B.1: An early Arduino board with an RS-232 serial interface and an Atmel ATmega8 microcontroller chip (Photo from https://en.wikipedia.org/wiki/File:Arduino316.jpg)

B.2 Arduino Software

Arduino programs may be written in any programming language with a compiler that produces binary machine code. Atmel provides a development environment for their microcontrollers, AVR Studio and the newer Atmel Studio.

The Arduino project provides the Arduino integrated development environment (IDE), which is a cross-platform application written in Java. It originated from the IDE for the Processing programming language project and the Wiring project. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and provides simple one-click mechanism for compiling and loading programs to an Arduino board. A program written with the IDE for Arduino is called a "sketch".

The Arduino IDE supports the C and C++ programming languages using special rules of code organization. The Arduino IDE supplies a software library called "Wiring" from the Wiring project, which provides many common input and output procedures. A typical Arduino C/C++ sketch consist of two functions that are compiled and linked with a program stub main() into an executable cyclic executive program:

- setup(): a function that runs once at the start of a program and that can initialize settings.
- loop(): a function called repeatedly until the board powers off.

After compilation and linking with the GNU toolchain, also including with the IDE distribution, the Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal coding that is loaded into the Arduino board by a loader program in the board's firmware.

B.3 Arduino Uno Board

(From Arduino official website https://www.arduino.cc/en/Main/ArduinoBoardUno Accessed: 12/12/2015)

The Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

"Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards see the Arduino index of boards.

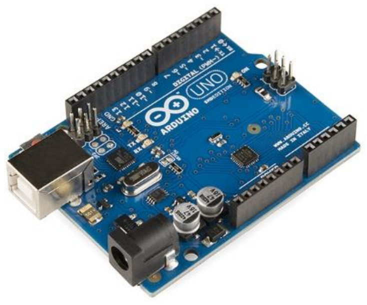


Figure B.2: The Arduino Uno Board (Photo from https://en.wikipedia.org/wiki/File:Arduino_Uno_-_R3.jpg)

<u>Board Technical Specification :</u>

- Microcontroller : ATmega328P
- Operating Voltage : 5V
- Input Voltage (recommended) : 7-12V
- Input Voltage (limit) : 6-20V
- Digital I/O Pins : 14 (of which 6 provide PWM output)
- Analog Input Pins : 6
- DC Current per I/O Pin : 20 mA
- DC Current for 3.3V Pin : 50 mA
- Flash Memory : 32 KB (ATmega328P) of which 0.5 KB used by bootloader
- SRAM : 2 KB (ATmega328P)
- EEPROM : 1 KB (ATmega328P)
- Clock Speed : 16 MHz
- Length : 68.6 mm
- Width : 53.4 mm
- Weight : 25 g

B.4 Arduino Nano Board

(From Arduino official website https://www.arduino.cc/en/Main/ArduinoBoardNano Accessed: 12/12/2015)

The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328 (Arduino Nano 3.x) or ATmega168 (Arduino Nano 2.x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. The Nano was designed and is being produced by Gravitech

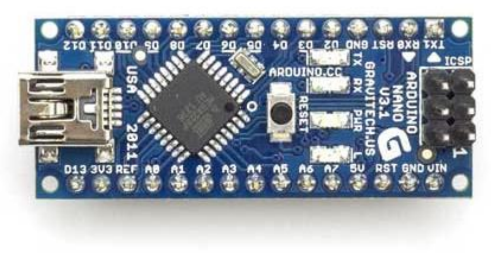

ARDUINO NANO V3.1
GRAVITECH.US
ARDUINO.CC
ICSP
RESET
USA
2011
D13 3V3 REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN
D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1

Figure B.3: The Arduino Nano Board (Photo from https://www.arduino.cc/en/uploads/Main/ArduinoNanoFront_3_lg.jpg)

<u>Board Technical Specification :</u>

- Microcontroller : Atmel ATmega168 or ATmega328
- Operating Voltage : 5V
- Input Voltage (recommended) : 7-12V
- Input Voltage (limit) : 6-20V
- Digital I/O Pins : 14 (of which 6 provide PWM output)
- Analog Input Pins : 6
- DC Current per I/O Pin : 40 mA
- Flash Memory : 16 KB (ATmega168) or 32 KB (ATmega328) of which 2 KB used by bootloader
- SRAM : 1 KB (ATmega168) or 2 KB (ATmega328)
- EEPROM : 512 bytes (ATmega168) or 1 KB (ATmega328)
- Clock Speed : 16 MHz
- Dimensions : 0.73" x 1.70"
- Length : 45 mm
- Width : 18 mm
- Weight : 5 g