Acknowledgments

We express our deep gratitude to our institute and management for providing us required funds, infrastructures, laboratory facilities, components, qualified and motivated staffs whose contribution make our project a successful one.

We would like to express our gratitude to our guide Dr. R. S. Geetha , Associate Professor, Department of Electrical and Electronics Engineering for her patience, immense knowledge, motivation and constant support. Her guidance has helped us throughout our project work.

Besides our guide, we would like to thank Prof. Harish V. Mekali, Associate Professor, Department of Electronics and Communication Engineering for his encouragement, insightful comments and constant support.

We would also like to thank Prof. Sreekanth N V, Assistant Professor, Departmenmt of Mechanical Engineering for his support in helping us with the 3-D printing process.

We convey our sincere gratitude to Dr. C. Lakshminarayana, Head of Department, Electrical and Electronics Engineering for encouraging and providing this opportunity to carry out the project in the department.

We are very thankful to Dr. Ravishankar Deekshit, Professor and Dean Student Affairs, Department of Electrical and Electronics Engineering for helping us obtain necessary funds and supporting us during every phase of the project.

We express our gratitude to our principal Dr. K. Mallikarjuna Babu, for providing an affable environment to work in.

We would like to thank TEQIP II for providing us funds for the second phase of the project.

We would also like to thank Dr. Nimish Soni, RBEI, for being very encouraging and supportive towards all of our requirements and aspirations.

Last but not the least we place a deep sense of gratitude to our family members and our friends who have been constant source of inspiration during the preparation of this project work.

Abstract

Robots are becoming indispensable in today's world. They are being used in a multitude of different sectors such as surgery, autonomous driving, etc. There is a growing requirement for intelligent robots that are capable of performing tasks that are time consuming or hazardous. Robots that will have an impact in industries or consumer markets will have to integrate several domains of robotics and coordinate these effectively.

ARDOP (Autonomous Robot Development Open-source Platform) is an open source humanoid robot that would serve as an extensive platform for the development of robotic applications. It is an economical platform that aims to make robots more commonplace. ARDOP could be 3D printed and is designed to be cost effective. The aim is to make a full-fledged integrated hardware and software platform so that it can be used for commercial or research purposes to accelerate the growth of robotics.

ARDOP possesses simple yet dextrous arms which enables flexibility while interacting with objects. It also incorporates elements that would enable better human interaction. The platform integrates various aspects of robotics and packages it into a single bundle which is easy to use and execute.

Using ARDOP, students, researchers and consumers will be able to use pre-made algorithms, modify them and execute them with ease. The different abstraction layers in the project enable people with all levels of technical expertise to build and develop algorithms and functionality in the robot.

List of Figures

List of Tables

Contents

Chapter 1

INTRODUCTION

The main intention of this project is to build an intelligent autonomous robot that would be used to forward research in the fields of computer vision, control, kinematics, motion planning and machine learning. The integration of these concepts is essential for any modern intelligent robot that wishes to make a mark in the rapidly advancing field of robotics. ARDOP would serve as a platform for students and researchers to implement their ideas and algorithms. Being economical, the platform would help make robotics research more feasible.

One of the primary aspects of ARDOP is its ability to perceive and interact with objects in its surroundings. Such a task is achieved by the seamless integration of various domains of robotics. Other open source robotics platforms do not include such a deep and extensive compilation of different concepts of intelligent robotics.

The entire design for the upper body of the humanoid robot has been created and tested using SolidWorks. The forward and inverse kinematic models for the arms of the robot have been designed and verified. This allows the control of the exact position and configuration of the arm in 3D space. The Jacobian Pseudoinverse iterative solution has been used to compute the inverse kinematics solutions. All the modules of the robot are implemented on the NVidia Jetson TK1 board.

An object localization and classification system has been built for the purpose of interaction. Localization is carried out using contour based computer vision algorithms. Convolution Neural Networks are used to train and classify the images of objects. The 3D coordinates of the objects in the scene are obtained using a Stereo Camera (DUO MLX) and this enables the robot to identify and interact with required objects in the scene. Appropriate trajectory planning algorithms are used to ensure that the arm configurations do not interfere and collide with other objects in the scene.

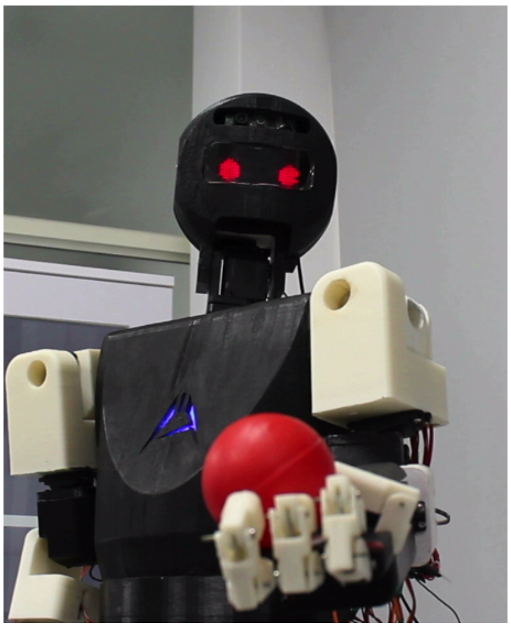


Figure 1.1: ARDOP

Chapter 2

LITERATURE SURVEY

As ARDOP integrates various systems in it's application, the literature survey extends through various fields. In this section, the advancements in each area and the integration of these areas in the field of robotics are described.

2.1 OBJECT RECOGNITION

The object recognition problem is solved in a plethora of ways today. Computer vision based methods are moving closer into the field of machine learning in order to get more accurate results. Classifiers such as neural networks and support vector machines are widely used to classify images and they tend to work well on images containing objects. Another popular method to find shapes in an image is the Hough transform [1]. This method has been extended in many ways in order to improve performance.

The main drawback in using a traditional machine learning approach is that the features have to be manually selected. Very often, researchers spend a major portion of their time finding appropriate features that describe the dataset. The performance of the system depends heavily on the features used as the input and the model would have to be implemented and trained to evaluate the contribution of the features in producing accurate results. This is very cumbersome and time consuming process and it significantly reduces productivity. Hence, computer vision algorithms are employed in order to extract features in the images which then can be fed into a classifier. A typical image classification process is shown in Figure 2.1

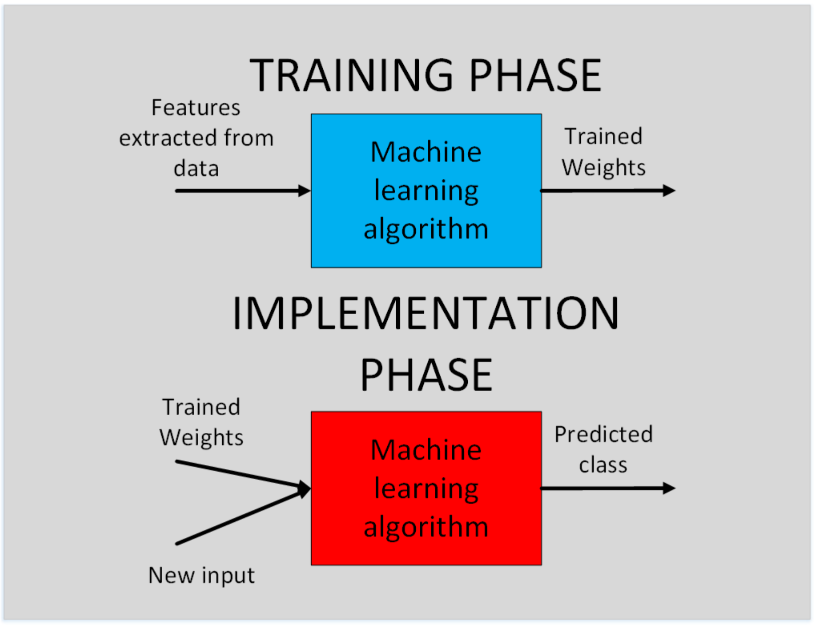


Figure 2.1: Image classification process

Popular method of object recognition used today is seen in [2, 3, 4]. Computer vision libraries such as OpenCV have inbuilt code written in order to implement these methods quickly and they provide a real-time recognition application. Although these algorithms work very fast and can provide high frame rates, they don't perform well on certain data sets.

Although first invented in the 1960s, Convolutional neural networks have made their way back into the modern scene as computational power has increased enormously in recent times. CNNs eliminate the problem of manual feature selection by learning appropriate filters which extract the most influential features for the given data. The paper [5] marked the revival of CNNs by winning the ImageNet challenge, that year by a large margin.

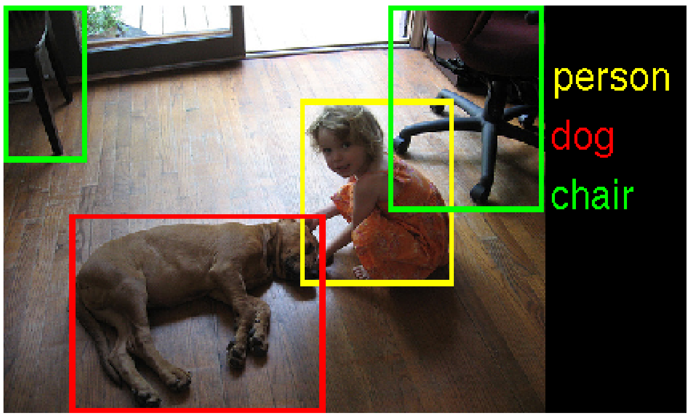


Figure 2.2: Example-Object Recognition

Source: ImageNet Dataset

2.2 KINEMATICS

Kinematics is a branch of mechanics that deals with the motion of objects without taking into account the forces that act upon them. Hence it is specifically used for pose estimation of objects-to find out their positions and orientations in three-dimensional space. Kinematics is used to study the motion of complex link-joint structures where each joint actuates a particular link system. By employing kinematic equations, it becomes possible to find out the joint parameters that would make the structure move in a way that is optimal for the given task. The end-effector of a robotic arm is the part that comprises of a gripper or some other manipulating structure which can be used to pick up objects, etc. The goal of the kinematic model is to enable the end-effector to reach the desired position to allow for interaction with real world objects. The solution obtained from the kinematic equations are the angle values for each of the joints that would make the end-effector move to the required position with the required orientation.

Kinematics is in turn divided into two parts-Forward and Inverse Kinematics. Forward

Kinematics deals with finding out the position and orientation of the end-effector, given the join values [6]. It is easy and straightforward to compute and is the first step towards building a kinematic model. Inverse Kinematics deals with computing the joint values in order for the end-effector to reach a certain position with a particular orientation. Inverse Kinematic solutions are highly non-linear and it becomes exceedingly difficult to compute for higher DOF systems [7].

The forward kinematics matrix is built using individual transformation matrices for the joints. A transformation matrix is used to pose of a link from one reference coordinate to the other. Thus, by multiplying all the transformation matrices of the individual joints, the pose of the end-effector with respect to the base coordinate frame- that is on the first link, can be calculated. The Forward Kinematics matrix is computed using the widely accepted "Denavit-Hartenberg" notation which requires each joint to be associated with a particular rectangular coordinate system. Finding the Inverse Kinematic solutions for joint angles is not as simple as finding the forward kinematics as it involves solving non-linear trigonometric equations, some of which have multiple solutions. The inverse kinematic solution can be obtained in a number of different ways, each having its own advantages and disadvantages.

Any system with more than 6 DOF will have a solution [8, 9], though it might be a redundant or singular solution. The condition for a system to have a solution for Inverse Kinematics is that it must possess 3 consecutive joints that are perpendicular to each other and have intersecting joint axes. If this condition is not satisfied, the joint angles for the problem cannot be found. Multiple redundant solutions are very likely for higher DOF systems and there is also a possibility for a singular position. A singular position is one wherein the system loses one or a few degrees of freedom due to other joint angles. An inverse kinematic solution becomes impossible in this condition. Mathematically, a singular position can be detected when the Jacobian of the Forward Kinematic Matrix loses its rank. Care must be taken to account for any singularities in the kinematic model [10].

Solving an Inverse Kinematics problem analytically means to compute the joint angles by manipulation of the Forward Kinematics equations [11]. An iterative method computes the joint angles by changing the joint angles by small amounts every iteration [12]. At one point, this process would converge and the joint angles would remain constant. The values at this point are taken to be the solution of the Inverse Kinematic problem. There are several iterative methods used to find the Inverse Kinematic solutions such as Cyclic Coordinate Descent, Jacobian Pseudoinverse, etc. Cyclic Coordinate Descent or CCD in short uses vector algebra to change the joint angles at every iteration [13]. Various position vectors are used to calculate the angle updates. In the Jacobian pseudoinverse method, the Jacobian of the Forward Kinematics Matrix is used to change the values of the joint angles at every step [14]. The Jacobian helps relate the Forward Kinematics Matrix to the angular velocities of the joints and is a very important tool in robot dynamics. In the iterative method, the Jacobian is approximated as a linear interpretation of the Forward Kinematics function and joint angle increments are substituted for the velocities. Thus, these joint angle changes can be computed by taking the inverse of the matrix.

2.3 PATH PLANNING

Path planning is the process of computing trajectories between two different configurations of a robot (Robotic arm in this case) [15, 16]. A time or iteration function is developed to help the arm to move from one orientation to another. While modelling such a function, the boundary conditions and other constraints are taken into account. The trajectories can be modelled taking into account only the rotational motion of the joints, or also the geometric curve between the two positions [17].

2.3.1 Joint Path

In joint path planning, only the joint variables or angles of the robotic arm are controlled. The exact trajectory of the end-effector is not taken into account. The only constraints for the position of the end-effector is that it is at the first position at the start of the motion and is at the final position once the motion is completed. The path that the end-effector traces out is not known.

When the robotic arm is allowed free motion and there are no obstacles in its workspace, then the trajectory that the robot takes is inconsequential. The arm can simply be made to move from one position to the other by simple joint control. The simplest form of joint control is a linear control where each of the joint angles are incremented or decremented so that the arm is able to move from one configuration to another. In such a model, the arm moves with a constant velocity and is prone to jerking (due to uncapped accelerations) at the start and end points of the motion [18]. As such a model only provides one trajectory per set of configurations, it is possible that the interpolation runs into positional singularities due to the design of the robotic arm. Such a planning method also provides no control over the velocities and accelerations of the arm which then makes it difficult to perform intricate tasks with consistency. To overcome the disadvantages of the linear control, other mathematical models are used which may be polynomial, harmonic, etc. Such models give more control over velocities and accelerations, while also giving a smoother overall motion. It is also possible to add more constraints such as positions, velocities and accelerations of the end-effector at certain times along the motion.

2.3.1.1 Linear Path

A linear joint trajectory uses linear equations to update the joint angles at specific time intervals. It is the simplest joint path planning construct as it involves only the basic constraints of initial and final configurations. A linear time dependent function is used to define the path. The joint equation is modelled as follows:

$$q(t) = at + b \tag{2.1}$$

q is the joint angle.

As the equation contains two unknowns, two known conditions are required to solve for the unknowns. The two conditions are the start and end configurations of the joint.

$$q(t_0) = q_0 \tag{2.2}$$

$$q(t_f) = q_f \tag{2.3}$$

t_0 is the starting time and q_0 is the initial joint angle.

t_f is the final time and q_f is the final joint angle.

More constraints cannot be added to such a model. No control can be exercised over the velocities or accelerations of the joints. Such a linear path is the bare minimum required to compute a joint trajectory between two configurations.

2.3.1.2 Cubic path

The most common type of joint trajectory is a cubic trajectory. A cubic equation has four unknowns and can thus calculate a function that can satisfy four constraints of boundary conditions. The most common conditions are initial joint angles, initial joint velocity, final joint angles and final joint velocity. These four conditions give four equations which can be used to find the cubic equation as a time function. The joint path is modelled as follows:

$$q(t) = at^3 + bt^2 + ct + d \tag{2.4}$$

q is the joint angle. As the equation contains four unknowns, four constraints can be added. The most common conditions that are added to the initial and final angle conditions are the initial and final joint velocities. The initial and final angle constraints are:

$$q(t_0) = q_0 \tag{2.5}$$
$$q(t_f) = q_f \tag{2.6}$$

To provide velocity constraints, the time differential of the joint equation must be taken and the required conditions must be plugged into this equation.

$$\dot{q}(t) = 3at^2 + 2bt + c \tag{2.7}$$

The velocity constraints may now be added as:

$$\dot{q}(t_0) = q'_0 \tag{2.8}$$
$$\dot{q}(t_f) = q'_f \tag{2.9}$$

q'_0 is the initial joint velocity and q'_f is the final joint velocity.

2.3.1.3 Piecewise path

A piecewise model can give more control to the trajectory. A very common requirement for robotic arms is for it to start from rest, acquire a velocity that remains constant till the end and then slowly go back to rest in the final position. Such a motion can be modelled by splitting the trajectory into three parts- rest to motion, constant velocity, motion to rest. Each of these pieces have their own boundary conditions and can be modelled using separate time functions. The first part is modelled as a quadratic equation and the last part is modelled as a cubic equation. The middle part of the motion is modelled as a linear time function (As the only condition is for the velocity to remain constant).

a) Rest to motion phase

   The initial rest to motion movement takes place from time t_0 to time t_1. In this phase, the initial position and velocity along with the final velocity at the start of the movement phase is known. Hence a quadratic time equation is used to define the joint path. The function along with its time derivative is as shown:

$$q_1(t) = at^2 + bt + c \tag{2.10}$$
$$\dot{q_1}(t) = 2at + b \tag{2.11}$$

This is subjected to the following constraints:

$$q_1(t_0) = q_0 \tag{2.12}$$
$$\dot{q_1}(t_0) = q_0' \tag{2.13}$$
$$\dot{q_1}(t_1) = q_c' \tag{2.14}$$

q_0' is the initial velocity right after the start of the motion and q_c' is the constant velocity of the joint.

b) Constant velocity phase

The constant velocity phase takes place from time t_1 to t_2. In this phase of motion, the joint is required to move at a constant velocity for a particular amount of time. This is the only constraint for this phase and is represented as shown:

$$\dot{q_2}(t) = q_c' \tag{2.15}$$

To obtain the joint equation, the constraint is integrated.

$$\int \dot{q}(t)dt = q_2(t) = q_c't + C \tag{2.16}$$

C is the constant of integration and is calculated by using a boundary condition. As the rest to motion phase and the constant velocity phase coincide at time t_1, the joint configuration at this time is the same in both the equations. Hence by equating the joint angle at time t_1, the value of the constant of integration can be found.

$$q_1(t_1) = q_2(t_1) \tag{2.17}$$
$$\Rightarrow at_1^2 + bt_1 + c = q_c't_1 + C \tag{2.18}$$
$$\Rightarrow C = at_1^2 + bt_1 + c - q_c't_1 \tag{2.19}$$

After C has been computed, the linear joint equation can be used to obtained the required rotation speed.

c) Motion to rest phase

The final phase of the piecewise trajectory is the motion to rest phase. It takes place from time t_2 to t_f. The initial and final position and velocities need to be satisfied, hence a cubic equation is used.

$$q_3(t) = at^3 + bt^2 + ct + d \tag{2.20}$$
$$\dot{q_3}(t) = 3at^2 + 2bt + c \tag{2.21}$$

The constraints used to find the equation parameters are:

$$q_3(t_2) = q_2(t_2) \tag{2.22}$$
$$\dot{q_3}(t_2) = q_c' \tag{2.23}$$
$$q_3(t_f) = q_f \tag{2.24}$$
$$\dot{q_3}(t_f) = q_f' \tag{2.25}$$
$$\tag{2.26}$$

q_f is the final joint angle and q_f' is the final joint angle velocity. Using these four constraints, the four unknowns of the cubic equation can be calculated and the joint time function for this phase of the motion is modelled . For the robotic arm to perform tasks such as picking up objects, more constraints might be needed. The robotic arm will have to approach the object through the direction best suitable for gripping. The velocities and jerks of the arm must be kept minimal as it approaches the object to prevent the arm from knocking the object off. In these cases, a lot more position and velocity constraints may be needed to be satisfied and thus higher degree polynomial equations may have to be used. This is because as the degree of the polynomial increases, more known conditions are needed to solve for the function parameters. For a trajectory with n constraints, an $n-1$ degree polynomial can be used to satisfy all the constraints. Another method of solving the time equations for such trajectories is to split the motion into multiple paths and then use linear or cubic equations to model each individual path [19].

2.3.2 Cartesian Path

In Cartesian Path Planning, the Cartesian coordinates of the end-effector are controlled [20]. In such a path planning construct, the entire trajectory of the arm is laid out in terms of the trajectory of the end effector. A time function of a three-dimensional curve in the Cartesian coordinate system is developed and the end effector of the arm is made to follow this curve. Thus, the position of the end-effector is well defined at every instant of the motion. No control is exercised over the angles or velocities of the individual joints. In Cartesian Path Planning, velocities and accelerations are defined with respect to the rate of change of the position of the end-effector.

Cartesian Path Planning is used when the approximate trajectory that the end-effector must take is known. This usually occurs when the positions of a few obstacles in the workspace of the robot are known. Sometimes, there are end-effector that may not be attainable from other configurations through minimal movement due to the presence of singularities while computing constrained Inverse Kinematics solutions. When there are forbidden points in the workspace, the trajectory that the arm takes must be designed or approximated around such points so that the whole motion remains stable and is within expected bounds. Cartesian Path Planning is thus essential when the robot has to perform tasks in a restricted work-space, subjected to physical constraints.

Apart from control over the Cartesian path taken by the end-effector, the positions and velocities of the end-effector at certain time instances may also be controlled by using higher degree polynomials to model the target time functions.

The most common type of path that connects two Cartesian coordinates is a straight-line path. Such a path connects both the targets using a straight-line equation. In the workspace of the robotic arm, the simplest trajectory to any given target is the straight-line path that connects the initial position of the end-effector to the target position. In some cases, such a straight-line path might contain unreachable points or singularities. When this happens, other valid points may be found in the workspace and the trajectory may be split into two straight-line motions. The region where both the trajectories transition may be smoothed out by an appropriate time function. Irrespective of how the motion is carried out, the straight-line path connecting two Cartesian targets is of utmost importance while planning Cartesian paths for the end-effector.

2.3.2.1 Straight line path with no additional constraints

The simplest straight line path between two coordinates use linear equations and only allow for two constraints. The constraints in this case are the starting and ending positions of the end-effector. The equations are described as follows:

$$x(t) = at + b \tag{2.27}$$
$$y(t) = ct + d \tag{2.28}$$
$$z(t) = et + f \tag{2.29}$$

The initial and final position constraints are as shown:

$$x(t_0) = x_0 \tag{2.30}$$
$$x(t_f) = x_f \tag{2.31}$$
$$y(t_0) = y_0 \tag{2.32}$$
$$y(t_f) = y_f \tag{2.33}$$
$$z(t_0) = z_0 \tag{2.34}$$
$$z(t_f) = z_f \tag{2.35}$$

The initial end-effector position is at $T_1 = (x_0, y_0, z_0)$ and the final end-effector position is at $T_2 = (x_f, y_f, z_f)$.

Using the six constraints, the six unknown equation parameters can be solved for and the straight-line trajectory between the two points is obtained. By plugging in time instances, target values at those instances are obtained and the Inverse Kinematics model is used to compute joint configurations for the end-effector to be positioned at that target coordinate.

2.3.2.2 Straight line path with quintic equations

If linear equations are used to model the straight-line trajectories, no other constraints such as velocities, accelerations, etc. can be placed on the end-effector. But for the robotic arm to perform intricate tasks, it might have to satisfy more constraints along with following a fixed cartesian path. When a robotic arm is designed to pick up objects, it is important for the arm to get to the location of the object slowly without any jerks. This can be achieved by setting the end velocity and acceleration to zero. Setting the initial velocity and acceleration to zero ensures that the motion is smooth and completely jerk free as a straight-line motion does not bring in unpredictable changes to acceleration.

Quintic equations cannot be used independently for all the three coordinate values as the resulting path would then not follow a straight-line. Thus, only one of the coordinates are modeled as an independent quintic equation and the other two coordinates are modeled with respect to the first coordinate using the straight-line slope equations. The x coordinate time function is modeled as a quintic function as shown:

$$x(t) = at^5 + bt^4 + ct^3 + dt^2 + et + f \tag{2.36}$$

The first and second time derivates are needed for velocity and acceleration constraint equations. They are as shown:

$$\dot{x}(t) = 5at^4 + 4bt^3 + 3ct^2 + 2dt + e \tag{2.37}$$

$$\ddot{x}(t) = 20at^3 + 12bt^2 + 6ct + 2d \tag{2.38}$$

The constraints to solve for the parameters are:

$$x(t_0) = x_0 \tag{2.39}$$
$$\dot{x}(t_0) = x'_0 \tag{2.40}$$
$$\ddot{x}(t_0) = x''_0 \tag{2.41}$$
$$x(t_f) = x_f \tag{2.42}$$
$$\dot{x}(t_f) = x'_f \tag{2.43}$$
$$\ddot{x}(t_f) = x''_f \tag{2.44}$$
$$\tag{2.45}$$

x_0, x'_0 and x''_0 are the initial position, velocity and acceleration of the x coordinate of the end-effector. x_f, x'_f and x''_f are the final position, velocity and acceleration of the x coordinate of the end-effector. These six constraints are then used to compute the equation parameters for the x coordinate. The y and z coordinate functions are then computed using the straight-line relations:

$$y(t) = y(T_1) + \frac{(y(T_2) - y(T_1))}{(x(T_2) - x(T_1))}(x(t) - x(T_1)) \tag{2.46}$$
$$z(t) = z(T_1) + \frac{(z(T_2) - z(T_1))}{(x(T_2) - x(T_1))}(x(t) - x(T_1)) \tag{2.47}$$

$x(T_1)$ and $x(T_2)$ are the x coordinate values at the positions T_1 and T_2 respectively. $y(T_1)$ and $y(T_2)$ are the x coordinate values at the positions T_1 and T_2 respectively. $z(T_1)$ and $z(T_2)$ are the x coordinate values at the positions T_1 and T_2 respectively. On computing the time functions for $x(t)$, $y(t)$ and $z(t)$, the time varying straight-line path of the end-effector is known. The Inverse Kinematic model is used to generate joint angles for these coordinates successively to realize the required arm trajectory.

2.4 STEREO VISION

Stereo vision is the process of extracting 3-D information from multiple 2-D views of a scene. The 3-D information is obtained from a pair of images, called as a stereo pair. By estimating the relative depth of points in the scene. These estimates are represented in a stereo disparity map, which is constructed by matching corresponding points in the stereo pair [21, 22]. In general, two cameras, displaced horizontally from one another are used to obtain two differing views on a scene. By comparing these two images, the relative depth information is obtained in the form of a disparity map, which maps the difference in horizontal coordinates of corresponding image points. The values in this disparity map are inversely proportional to the scene depth at the corresponding pixel location. The active stereo vision is a form of stereo vision which actively employs a light such as a laser or a structured light to simplify the stereo matching problem. The opposed term is passive stereo vision. This process is inspired by the biological process of how living beings perceive vision. Figure 2.3 represents the left and right images taken by a stereo camera system, the depth map obtained and the ground truth of the depth. Stereo vision is widely used to localize objects for manipulation tasks [23, 24].

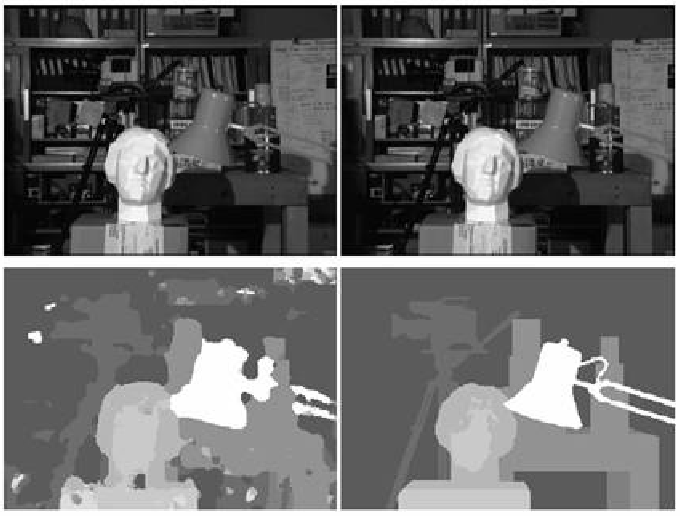


Figure 2.3: Example-Stereo Depth Calculation

Source: Middlebury Stereo Dataset

2.5 KALMAN FILTER

While measuring and controlling dynamic systems, it becomes necessary to account for all the small variations and errors that are caused due to inaccuracies in measurement and various environmental factors. Measurements would continuously vary which would cause changes in behaviour or accuracies of various models that require high levels of precision. Thus, an algorithm that will compute the most probably value of measurement from a set of measurements is required.

Kalman Filter [25] is an algorithm that uses series of measurements containing noises and inaccuracies, and produces estimates of unknown variables that tends to be more accurate, when compared to single measurement alone. The predicted value as computed by the Kalman Filter is the most expected value and the value with the least error. The filter is implemented by using Bayesian interference and estimating a joint probability distribution over the variables for each time frame.

Kalman filter is used in numerous applications such as guidance, navigation and control of vehicles in particular aircraft and spacecraft. It is widely applied concept used in signal processing and econometrics. It can be used in any place which requires the computation of the most expected value from a set of values [26]. A vectorized implementation of the filter is usually used which makes the complexity of computation lesser while computing the value in real-time. It also ensures that the model built is expandable to account for more variables that may or may not be correlated with each other.

1. Prediction step

   In this step, kalman filter produces estimates of the current state variables, along with their uncertainties [27]. Certain fixed values of correlation and variance for the measurement are used to compute the states of the variables. Thus, a rough estimate of the variance of the measurement is required to account for random noise and errors. The correlation matrix between the state variables is also constructed which tells the system about the dependence of the variables with each other.

2. Updating step

   Once the outcome of the next measurement is observed, these estimates are updated using a weighted average. Higher value of weights is assigned to estimates with higher certainty. Certain parameters and gains calculated in the previous step is used to update the value of the state variables. The value of the correlation matrix or error matrix is also updated which will be used for the prediction step in the next iteration.

The algorithm is recursive and can be run in real time, using only the present input, measurements and the previously calculated state and its uncertainty matrix. The algorithm uses only the knowledge regarding the expected or theoretical behaviour of the system, estimated measurement variances and current measurement of the state variables to compute the most expected value. The algorithm converges very quickly and attains a steady state value in a few iterations.

2.6 Emotion Display

Displaying human emotions on a robotic eye plays a significant role in expressing the mood of the robot. It also makes the communication with human beings more effective. Hence a proper method to convey the emotions of the robot is necessary to express the robot's mood. This can be done in two ways. One of them is by using direct prosthetic mechanism and the other is by using led matrices. Even though the results in prosthetic mechanism are extremely good, it turns out to be a very expensive implementation. On the other hand, led matrices give good results and turn out to be extremely cost effective. Hence we have chosen to implement the robotic emotions via eyes by using a pair of custom made led matrix.

Chapter 3

AIM AND SCOPE

The aim of the project is to develop an economical and versatile open source platform for the development of robotic applications. The platform would encompass both hardware and software modules which can be modified according to specific needs. The project aims to build an extensive model invariant cognitive platform which would help users share their learning systems effectively. Functionality such as facial and object recognition, object interaction, etc. would be provided in the form of black-boxed functions which can directly be used for development of other features.

The platform can be used either for educational purposes to help students and researchers test and develop new algorithms in the field of robotics. Being economical, it would forward the development of such research by making humanoids more accessible. The robot can also be used in places such as hospitals, restaurants, old age homes, etc. to automate certain processes or to monitor specific events. ARDOP will be equipped with a simple interface that would enable any individual to use the robot to complete specific tasks.

Chapter 4

METHODOLOGY AND IMPLEMENTATION

Object recognition involves the classification and hence recognition of different objects. For a robot to interact with objects, it first needs to identify all the objects in the scene and localize its location in the environment. Convolution Neural Networks are used to implement such a recognition system.

Kinematics is necessary to enable the arms of the robot to move to specific points in 3D space. This is essential for the robot to interact with objects in its environment.

Path planning involves the computation of joint or end-effector trajectories to obtain the required arm trajectories. Path and collision constraints are controlled using such algorithms.

The motors of ARDOP are controlled using i2c using an i2c controller. The controller translates PWM signals to the servo motors to control their rotation.

Stereo vision involves the computation of depth from image frames. A stereo camera is used to obtain the depth map of the camera feed. The objects in the scene can thus be localized using the stereo camera.

Kalman Filter is an algorithm that computes the most probable or expected value of a set of measurements. It uses a probabilistic algorithm to achieve this. The filter is used to reduce the error from distance measurement.

Object interaction basically involves picking up, dropping and manipulation of objects. It is an integration of the other domains explained above.

An emotion display system is built for effective human robot interaction. Various expressions are coded into the system for use in different situations.

4.1 OBJECT RECOGNITION

4.1.1 Convolutional Neural Networks

The problem with regular fully connected neural networks is that with larger images (e.g., 96x96 images) learning features that span the entire image (fully connected networks) is very computationally expensive.

One simple solution to this problem is to restrict the connections between the hidden units and the input units, allowing each hidden unit to connect to only a small subset of the input units. Specifically, each hidden unit will connect to only a small contiguous region of pixels in the input. This idea of having locally connected networks also draws inspiration from how the early visual system is wired up in biology. Specifically, neurons in the visual cortex have localized receptive fields (i.e., they respond only to stimuli in a certain location).

4.1.1.1 Convolution Layer

Natural images have the property of being "stationary", meaning that the statistics of one part of the image are the same as any other part. This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations. Having learned features over small patches sampled randomly from the larger image, we can then apply this learned feature detector anywhere in the image. Given an image, not only are the weights between the layers trained (like in a fully connected neural network), but the filters that extract features from the image are also learned by the network. This results in the model learning the best features that allow the network to accurately classify an image.

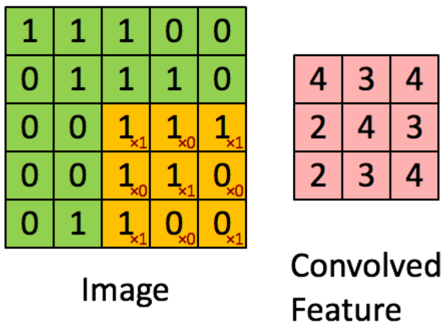


Figure 4.1: Example- Convolution

4.1.1.2 Pooling Layer

After obtaining features extracted using convolution, a classifier could be trained using all the features, but would be computationally expensive. A classifier having a large number of inputs could also be prone to over-fitting and this is undesirable. To address this issue, the image is made into sections and the statistics of the various features in the sections are aggregated. The mean or max value in these sections could be computed and this would result in a dimensionality reduction, thus overcoming over-fitting. This process is called pooling (max-pooling or mean-pooling depending on the operation used).

4.1.1.3 Fully Connected Layer

The term fully connected implies that every neuron in the previous layer is connected to every neuron in the next layer. Fully connected layers are used to use the features extracted by the other layers to classify the given images. This layer also learns a non-linear combination of these features which may results in better classification.

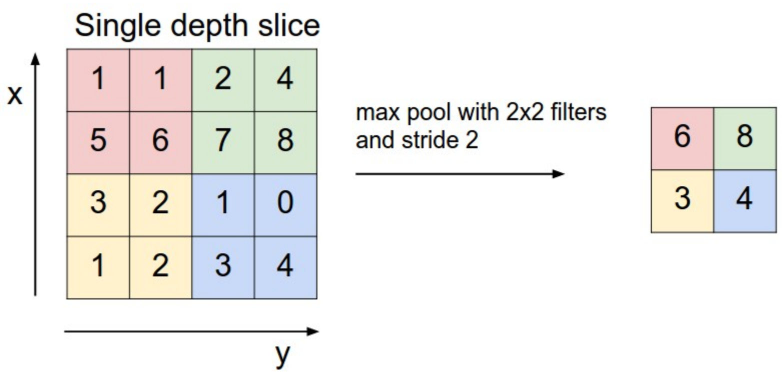


Figure 4.2: Example- Pooling

4.1.1.4 Architecture

The architecture of a CNN greatly influences the accuracy of the model. A CNN consists of a number of convolutional and sub-sampling layers which is then followed by fully connected. The input to a convolutional layer would be an m x m x r image, m being the height and width and r being the number of channels. The convolutional layer would have k filters each of size n x n x q where n is smaller than the image size. The output of each layer is connected to the next in order to perform an operation and carry forward the inputs through the network. On arranging these layers accordingly, the appropriate features that describe the data can be extracted and fed to the fully connected layer to predict the given image.

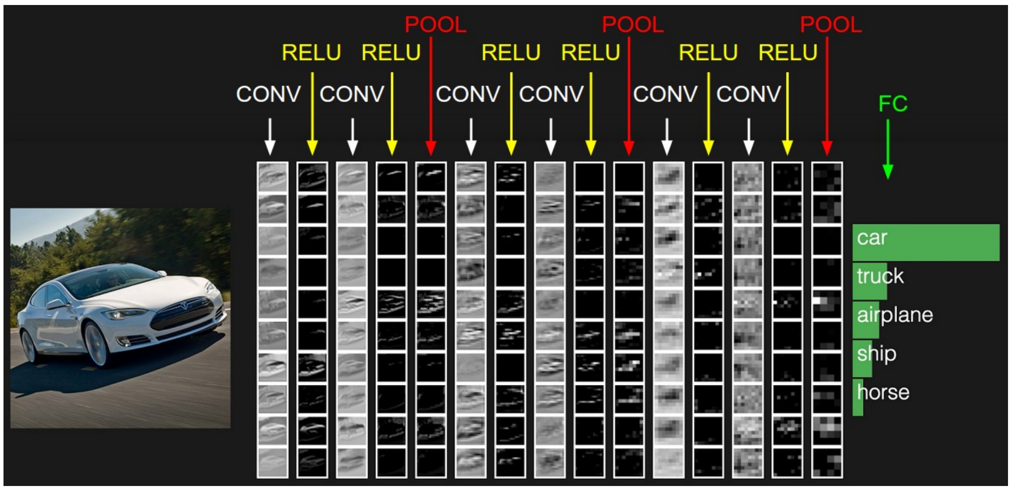
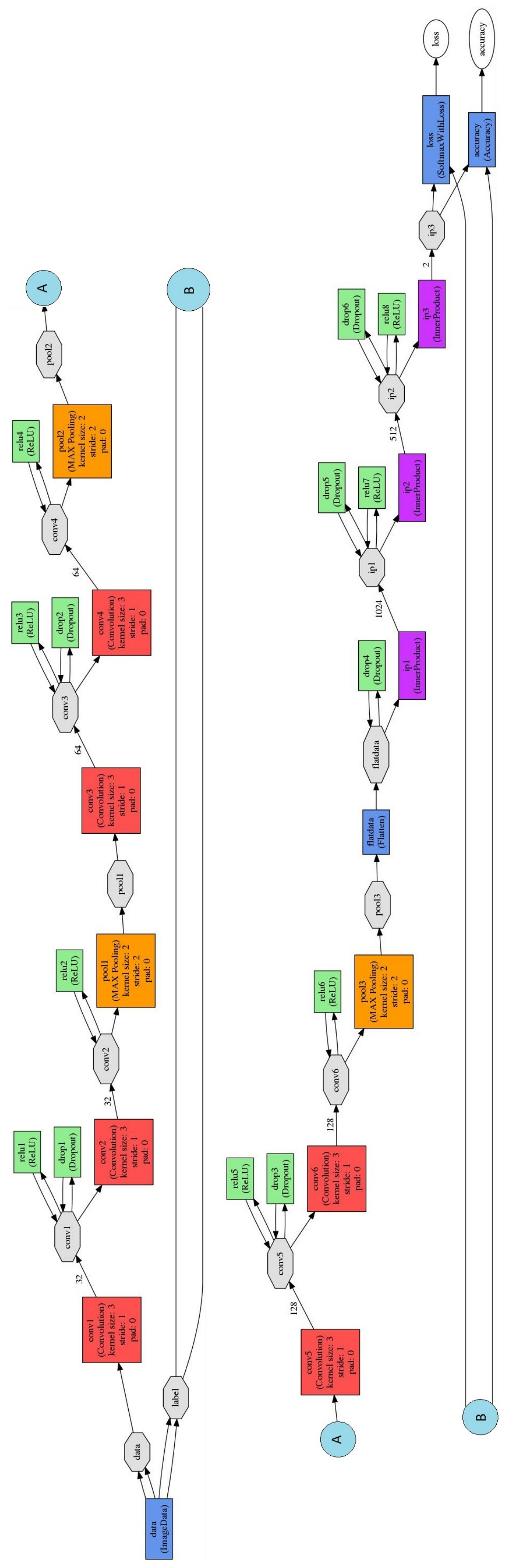


Figure 4.3: Typical convolutional neural network

Source: Stanford CS231n

4.1.2 Implementation

The CNN was implemented using caffe [28], a highly optimized library used for deep learning. The layers and architecture were defined as seen in Figure 4.4. Two objects- a glass and a ball were trained by building a dataset of 198 images for training and 60 images for testing. The solver was defined using a base learning rate of 0.001 and momentum 0.9.



data (ImageData)
data
label
conv1 (Convolution) kernel size: 3 stride: 1 pad: 0
32
conv1
relu1 (ReLU)
drop1 (Dropout)
conv2 (Convolution) kernel size: 3 stride: 1 pad: 0
32
conv2
relu2 (ReLU)
pool1 (MAX Pooling) kernel size: 2 stride: 2 pad: 0
pool1
conv3 (Convolution) kernel size: 3 stride: 1 pad: 0
64
conv3
relu3 (ReLU)
drop2 (Dropout)
conv4 (Convolution) kernel size: 3 stride: 1 pad: 0
64
conv4
relu4 (ReLU)
pool2 (MAX Pooling) kernel size: 2 stride: 2 pad: 0
pool2
A
B
A
conv5 (Convolution) kernel size: 3 stride: 1 pad: 0
128
conv5
relu5 (ReLU)
drop3 (Dropout)
conv6 (Convolution) kernel size: 3 stride: 1 pad: 0
128
conv6
relu6 (ReLU)
pool3 (MAX Pooling) kernel size: 2 stride: 2 pad: 0
pool3
flatdata (Flatten)
flatdata
drop4 (Dropout)
ip1 (InnerProduct)
1024
ip1
drop5 (Dropout)
relu7 (ReLU)
ip2 (InnerProduct)
512
ip2
drop6 (Dropout)
relu8 (ReLU)
ip3 (InnerProduct)
2
ip3
loss (SoftmaxWithLoss)
loss
accuracy (Accuracy)
accuracy
B

Figure 4.4: CNN architecture used for classification

4.2 KINEMATICS

ARDOP has two arms, each with five degrees of freedom (DOF). DOF is an important term in kinematics and it refers to the total number of independent aspects of motion. Each arm comprises of five servo motors which in turn implies five DOF [29]. In kinematics, each DOF is represented using a single variable (θ) which represents the angle for that particular joint. Hence, five such angle values are solved using the kinematic equations and the servo motors of the arm are moved to those angles.

4.2.1 Forward kinematics

The joint coordinates are chosen in such a way that the same transformation matrix can be used for all the joints, with changes made to the joint parameters (angle and length). The coordinates are also chosen in such that the same model can be used for both the arms. The transformation matrix used can be represented as follows:

$$T_i^{i-1} = \begin{bmatrix} cos\theta_i & 0 & sin\theta_i & 0 \\ sin\theta_i & 0 & -cos\theta_i & 0 \\ 0 & 1 & 0 & l_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

The Forward Kinematics matrix is computed using:

$$T_5^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

The end-effector orientation is given by ‘n’, ‘o’ and ‘a’ in terms of some Euler rotation matrix and the Cartesian position is given by ‘p’. The matrix thus encodes the orientation and positional information of the end-effector with reference to the base frame.

F is a function of the joint parameters and can hence be calculated by multiplying the five transformation matrices and substituting the required angles of the joints.

After calculation, the following values for the Forward Kinematics Matrix is obtained:

$$n_x = cos\theta_1 cos\theta_2 cos\theta_3 cos\theta_4 cos\theta_5 + cos\theta_1 cos\theta_2 sin\theta_3 sin\theta_5 + cos\theta_1 cos\theta_5 sin\theta_2 sin\theta_4 + cos\theta_4 cos\theta_5 sin\theta_1 sin\theta_3 - cos\theta_3 sin\theta_1 sin\theta_5 \tag{4.3}$$

$$o_x = cos\theta_1 cos\theta_2 cos\theta_3 sin\theta_4 - cos\theta_1 cos\theta_4 sin\theta_2 + sin\theta_1 sin\theta_3 sin\theta_4 \tag{4.4}$$

$$a_x = cos\theta_1 cos\theta_2 cos\theta_3 cos\theta_4 cos\theta_5 - cos\theta_1 cos\theta_2 cos\theta_5 sin\theta_3 + cos\theta_1 sin\theta_2 sin\theta_4 sin\theta_5 + cos\theta_4 sin\theta_1 sin\theta_3 sin\theta_5 + cos\theta_3 cos\theta_5 sin\theta_1 \tag{4.5}$$

$$p_x = l_5 cos\theta_1 cos\theta_2 cos\theta_3 sin\theta_4 - l_5 cos\theta_1 cos\theta_4 sin\theta_2 - l_3 cos\theta_1 sin\theta_2 + l_5 sin\theta_1 sin\theta_3 sin\theta_4 - l_2 sin\theta_1 \tag{4.6}$$

$$n_y = cos\theta_1 cos\theta_2 cos\theta_3 cos\theta_4 sin\theta_1 + cos\theta_1 sin\theta_1 sin\theta_3 sin\theta_5 + cos\theta_5 sin\theta_1 sin\theta_2 sin\theta_4 - cos\theta_1 cos\theta_5 sin\theta_3 sin\theta_4 + cos\theta_1 cos\theta_3 sin\theta_5 \tag{4.7}$$

$$o_y = cos\theta_2 cos\theta_3 sin\theta_1 sin\theta_4 - cos\theta_4 sin\theta_1 sin\theta_2 - cos\theta_1 sin\theta_3 sin\theta_4 \tag{4.8}$$

$$a_y = cos\theta_2 cos\theta_3 cos\theta_4 sin\theta_1 sin\theta_5 - cos\theta_2 cos\theta_5 sin\theta_1 sin\theta_3 + sin\theta_1 sin\theta_2 sin\theta_4 sin\theta_5 - cos\theta_1 cos\theta_4 sin\theta_3 sin\theta_5 - cos\theta_1 cos\theta_3 cos\theta_5 \tag{4.9}$$

$$p_y = l_5 cos\theta_2 cos\theta_3 sin\theta_1 sin\theta_4 - l_5 cos\theta_4 sin\theta_1 sin\theta_2 - l_3 sin\theta_1 sin\theta_2 - l_5 cos\theta_1 sin\theta_3 sin\theta_4 + l_2 cos\theta_1 \tag{4.10}$$

$$n_z = cos\theta_3 cos\theta_4 cos\theta_5 sin\theta_2 + sin\theta_2 sin\theta_3 sin\theta_5 - cos\theta_2 cos\theta_5 sin\theta_4 \tag{4.11}$$

$$o_z = cos\theta_3 sin\theta_2 sin\theta_4 + cos\theta_2 cos\theta_4 \tag{4.12}$$

$$a_z = cos\theta_3 cos\theta_4 sin\theta_2 sin\theta_5 - cos\theta_5 sin\theta_2 sin\theta_3 - cos\theta_2 sin\theta_4 sin\theta_5 \tag{4.13}$$

$$p_z = l_5 cos\theta_3 sin\theta_2 sin\theta_4 + l_5 cos\theta_2 cos\theta_4 + l_3 cos\theta_2 + l_1 \tag{4.14}$$

4.2.2 Inverse kinematics

For ARDOP, The Jacobian Pseudoinverse method is used to obtain the Inverse Kinematics solutions.

i) Inverse Kinematics using Jacobian Pseudoinverse The equations used for Inverse Kinematics using the Jacobian pseudoinverse is as shown:

$$J(\theta) = (\frac{\partial F_k}{\partial \theta_j})_k, j \tag{4.15}$$

$$\Delta\theta = J^{\dagger}\Delta t \tag{4.16}$$

$$\theta = \theta \pm \Delta\theta \tag{4.17}$$

$$error = \sqrt{\Sigma(t_{required} - t_{calculated})^2} \tag{4.18}$$

The algorithm used to compute the solution is as shown:

1. The initial θ value is initialized randomly. Iter $\leftarrow$ 0.
2. For i=1 to 500 do steps 3-5
3. The Jacobian and its pseudoinverse is computed using this θ
4. The input value for the target position is used to compute Δt.
5. The theta update is calculated and theta is updated.
6. The end effector position is calculated using the updated θ.
7. The error between the required target and the calculated target is computed.
8. If the error is less than the threshold and the calculated angles are valid, the solution has been found and the program ends.
9. Iter $\leftarrow$ Iter + 1
10. If Iter > 100, No solution is possible
11. Else, go to step 1.

The solution computed by this method depend on the initial angle values. Thus, different solutions are obtained for different initial values. Thus, if a good solution is not obtained (Due to inaccuracy or validity), the initial angles are re-computed randomly and the program is run again.

$J^{\dagger}$ is the Jacobian pseudoinverse and Δ t is the difference between the current end-effector position vector and the target vector.

This method allows for fast computation of the joint angles as the joint updates depend on intrinsic joint parameters. It is a very popular method for computing Inverse Kinematic solutions of highly redundant robotic systems.

The Jacobian Pseudoinverse method is affected by any singularities present in the arm. If such singularities are encountered, the system would fail to converge and the system would be thrown out of stability. Thus, care must be taken while employing this method for higher DOF singularity prone arms.

4.3 PATH PLANNING

In any path planning construct, the degree of the time equation depends on the number of constraints that need to be satisfied. The degree of the equation or the structure of the model is chosen depending on these constraints. Hence the number of equations generated from the time equation will always be equal to the number of constraints.

The parameters that need to be computed to solve the path planning problem are the constants of the modelling functions. Each function can be thought of as a linear combination of these parameters. The constants that are multiplied with these parameters are the powers of time t.

The linear equations may be represented as simple matrix multiplications and solved by taking inverses of the matrices. As the time instances represented by t is non-repeating and the polynomial functions are continuous, all the rows of the time matrix are linearly independent. This ensures that the matrix is invertible and a solution is always possible for well-behaved constraints.

4.3.1 Joint Path

4.3.1.1 Linear Path

Linear equations are used to represent the joint time function. The function used is:

$$q(t) = at + b \tag{4.19}$$

$$\tag{4.20}$$

It is subjected to the conditions:

$$q(t_0) = q_0 \tag{4.21}$$

$$q(t_f) = q_f \tag{4.22}$$

Hence the two equations obtained are:

$$at_0 + b = q_0 \tag{4.23}$$

$$at_f + b = q_f \tag{4.24}$$

Using these two equations, the two unknowns a and b can be solved. To generalize the process for higher degree polynomials and different path planning algorithms, the equations are represented in the form of a matrix equation. In the matrix form:

$$\begin{bmatrix} t_1 & 1 \\ t_f & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} q_0 \\ q_f \end{bmatrix} \tag{4.25}$$

If A=$\begin{bmatrix} t_1 & 1 \\ t_f & 1 \end{bmatrix}$, B=$\begin{bmatrix} q_0 \\ q_f \end{bmatrix}$ and x=$\begin{bmatrix} a \\ b \end{bmatrix}$, The equation takes the form:

$$Ax = B \tag{4.26}$$

The function parameters are then calculated using:

$$x = A^{-1}B \tag{4.27}$$

As ARDOP has five joints, these equations must be generated for all the five joints, using the initial and final configurations of all the joints. Instead of computing them separately for the joints, all the parameters are incorporated into the same matrix equation. Now the i^{th} joint has function parameters a_i and b_i and initial and final configurations equal to q_0^i and q_f^i

$$\begin{bmatrix} t_0 & 1 \\ t_f & 1 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \\ b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} = \begin{bmatrix} q_0^1 & q_0^2 & q_0^3 & q_0^4 & q_0^5 \\ q_f^1 & q_f^2 & q_f^3 & q_f^4 & q_f^5 \end{bmatrix} \tag{4.28}$$

The parameter matrix x is solved using the same way using the inverse of the time matrix.

4.3.1.2 Cubic path

The time function and its time derivative used is:

$$q(t) = at^3 + bt^2 + ct + d \tag{4.29}$$

$$\dot{q}(t) = 3at^2 + 2bt + c \tag{4.30}$$

In the combined matrix form for all angles,

$$\begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 \\ 3t_0^2 & 2t_0 & 1 & 0 \\ t_f^3 & t_f^2 & t & 1 \\ 3t_f^2 & 2t_f & 1 & 0 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \\ b_1 & b_2 & b_3 & b_4 & b_5 \\ c_1 & c_2 & c_3 & c_4 & c_5 \\ d_1 & d_2 & d_3 & d_4 & d_5 \end{bmatrix} = \begin{bmatrix} q_0^1 & q_0^2 & q_0^3 & q_0^4 & q_0^5 \\ q_0^{1'} & q_0^{2'} & q_0^{3'} & q_0^{4'} & q_0^{5'} \\ q_f^1 & q_f^2 & q_f^3 & q_f^4 & q_f^5 \\ q_f^{1'} & q_f^{2'} & q_f^{3'} & q_f^{4'} & q_f^{5'} \end{bmatrix} \tag{4.31}$$

Or in the shorthand form:

$$Ax = B$$

Which is solved using

$$x = A^{-1}B \tag{4.32}$$

4.3.1.3 Piecewise Path

(A) REST TO MOTION PHASE

For a piecewise trajectory encompassing a constant velocity phase, a quadratic joint model is generally used.

$$q_1(t) = at^2 + bt + c \tag{4.33}$$

$$\dot{q_1}(t) = 2at + b \tag{4.34}$$

Such an equation would work really well for planar robotic structures having less Degrees of Freedom. As the number of Degrees of Freedom increase and the number of perpendicular joints in the robotic arm increases, the stability of such an equation reduces as the direction of motion becomes more uncertain. As such a quadratic equation cannot satisfy an end configuration condition, for complex robotic structures, there is a lot of doubt as to the direction the trajectory should take given the starting configuration and velocity.

Such a quadratic system has been tested on ARDOP and the results obtained were not satisfactory. To overcome this difficulty, a few extra implementation steps have been added. First, apart from the initial and final configurations, two new configurations at t_1 and t_2 (point of motion transition) are computed. This is done by computing the parameters of a cubic equation as described in the previous section. The configurations at the required time intervals are then calculated by inputting the time instances to the cubic function. The two computed configurations are q_1^i and q_2^i for the i^th joint.

The rest to motion phase is now modelled as a cubic equation with initial and final configuration and velocity constraints. The following equation is solved to obtain the function parameters:

$$\begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 \\ 3t_0^2 & 2t_0 & 1 & 0 \\ t_f^3 & t_f^2 & t & 1 \\ 3t_f^2 & 2t_f & 1 & 0 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \\ b_1 & b_2 & b_3 & b_4 & b_5 \\ c_1 & c_2 & c_3 & c_4 & c_5 \\ d_1 & d_2 & d_3 & d_4 & d_5 \end{bmatrix} = \begin{bmatrix} q_0^1 & q_0^2 & q_0^3 & q_0^4 & q_0^5 \\ q_0^{1'} & q_0^{2'} & q_0^{3'} & q_0^{4'} & q_0^{5'} \\ q_1^1 & q_1^2 & q_1^3 & q_1^4 & q_1^5 \\ q_1^{1'} & q_1^{2'} & q_1^{3'} & q_1^{4'} & q_1^{5'} \end{bmatrix} \tag{4.35}$$

(B) CONSTANT VELOCITY PHASE

As ARDOP has 5 Degrees of Freedom with multiple perpendicular joints, the standard model for the piecewise constant velocity trajectory does not work well. If a given constant velocity is given as a constraint, the three phases of the motion tend to fall apart and become unstable. This is because, for complex structures, it is not possible to have each joint move at the same velocity and follow the expected trajectory. Moreover, the rest to motion phase equation has been updated to incorporate the newly computed configurations. This renders the standard constant velocity phase model unsuitable with the previous phase. To fix this, the constant velocity is computed using q_1^i and q_2^i and this computed velocity is used as a constraint. The velocity is calculated in the following way:

$$q_2(t) = q_c' t + C \tag{4.36}$$

q_1^i and q_2^i must be configurations that coincide with the motion of the constant velocity phase. They are the configurations at time t_1 and t_2.

$$q_2(t_1) = q_1 \tag{4.37}$$

$$q_2(t_1) = q_2 \tag{4.38}$$

$$\implies q_c' t_1 + C = q_1 \tag{4.39}$$

$$\implies q_c' t_2 + C = q_2 \tag{4.40}$$

$$\implies C = q_c' t_1 - q_1 = q_c' t_2 - q_2 \tag{4.41}$$

$$\implies q_c' = \frac{(q_1 - q_2)}{(t_1 - t_2)} \tag{4.42}$$

If the velocity is calculated this way instead of giving any arbitrary value, it ensures that the constant velocity phase is achieved and the transition between the three phases is perfect. To calculate the constant velocities of each joint, the following matrices are used for q_c', q_1 and q_2.

$$q'_c = \begin{bmatrix} q_c^{1'} & q_c^{2'} & q_c^{3'} & q_c^{4'} & q_c^{5'} \end{bmatrix} \tag{4.43}$$

$$q_1 = \begin{bmatrix} q_1^1 & q_1^2 & q_1^3 & q_1^4 & q_1^5 \end{bmatrix} \tag{4.44}$$

$$q_2 = \begin{bmatrix} q_2^1 & q_2^2 & q_2^3 & q_2^4 & q_2^5 \end{bmatrix} \tag{4.45}$$

The constant of integration C for all the joint functions can now be found as:

$$C = q'_c t_1 - q_1 \tag{4.46}$$

Now the function parameter matrix is found as: Now the function parameter matrix is found as:

$$x = \begin{bmatrix} q'_c \\ C \end{bmatrix} \tag{4.47}$$

(C) MOTION TO REST PHASE

There are no changes to the standard cubic function for the motion to rest phase. This is because it already accounts for four constraints which are the initial and final configurations and velocities. In this case, the initial time is at t_2. The following cubic equation and its time derivative is used to model the joints in this phase:

$$q_3(t) = at^3 + bt^2 + ct + d \tag{4.48}$$

$$\dot{q_3}(t) = 3at^2 + 2bt + c \tag{4.49}$$

In the matrix form:

$$\begin{bmatrix} t_2^3 & t_2^2 & t_2 & 1 \\ 3t_2^2 & 2t_2 & 1 & 0 \\ t_f^3 & t_f^2 & t_f & 1 \\ 3t_f^2 & 2t_f & 1 & 0 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \\ b_1 & b_2 & b_3 & b_4 & b_5 \\ c_1 & c_2 & c_3 & c_4 & c_5 \\ d_1 & d_2 & d_3 & d_4 & d_5 \end{bmatrix} = \begin{bmatrix} q_2^1 & q_2^2 & q_2^3 & q_2^4 & q_2^5 \\ q_2^{1'} & q_2^{2'} & q_2^{3'} & q_2^{4'} & q_2^{5'} \\ q_f^1 & q_f^2 & q_f^3 & q_f^4 & q_f^5 \\ q_f^{1'} & q_f^{2'} & q_f^{3'} & q_f^{4'} & q_f^{5'} \end{bmatrix} \tag{4.50}$$

Or:

$$Ax = B \tag{4.51}$$

Which is solved using

$$x = A^{-1}B \tag{4.52}$$

Once the function parameter matrix is known, the entire time dependent function is well defined and by plugging in values of specific time instances, the function outputs the configuration of the arm or the angles of all the five joints. This process is done continuously and at each iteration, the motors in each of the joints are rotated to the computed angles. This completes the required trajectory from one configuration to another.

4.3.2 Cartesian Path

4.3.2.1 Straight-line path with no additional constraints

The equations used to describe the cartesian path are:

$$x(t) = at + b \tag{4.53}$$

$$y(t) = ct + d \tag{4.54}$$

$$z(t) = et + f \tag{4.55}$$

Subjected to the constraints, the following equations are obtained:

$$x(t_0) = at_0 + b = x_0 \tag{4.56}$$

$$x(t_f) = at_f + b = x_f \tag{4.57}$$

$$y(t_0) = ct_0 + d = y_0 \tag{4.58}$$

$$y(t_f) = ct_f + d = y_f \tag{4.59}$$

$$z(t_0) = et_0 + f = z_0 \tag{4.60}$$

$$z(t_f) = et_f + f = z_f \tag{4.61}$$

These six equations can be used to solve for the function parameters a, b, c, d, e and f. In the matrix form, the equations can be written as:

$$\begin{bmatrix} t_0 & 1 \\ t_f & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x_0 \\ x_f \end{bmatrix} \tag{4.62}$$

$$\begin{bmatrix} t_0 & 1 \\ t_f & 1 \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} y_0 \\ y_f \end{bmatrix} \tag{4.63}$$

$$\begin{bmatrix} t_0 & 1 \\ t_f & 1 \end{bmatrix} \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} z_0 \\ z_f \end{bmatrix} \tag{4.64}$$

The combined matrix equation for all angles for each coordinate is represented as:

$$\begin{bmatrix} t_0 & 1 \\ t_f & 1 \end{bmatrix} \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & z_0 \\ x_f & y_f & z_f \end{bmatrix} \tag{4.65}$$

a_i and b_i are the function parameters for the x coordinate for the i^th joint angle.x_i^j is the x coordinate cartesian value of the end-effector at time t_i for thej^thjoint.

If they can be represented in the shorthand notation as:

$$Ax = B_x \tag{4.66}$$

$$Ay = B_y \tag{4.67}$$

$$Az = B_z \tag{4.68}$$

Then they are solved by taking the inverse of the time matrix A

$$x = A^{-1}B_x \tag{4.69}$$

$$y = A^{-1}B_y \tag{4.70}$$

$$z = A^{-1}B_z \tag{4.71}$$

4.3.2.2 Straight-line path using a quintic equation

The x coordinate is modelled using a quintic equation as shown:

$$x(t) = at^5 + bt^4 + ct^3 + dt^2 + et + f \tag{4.72}$$

The combined matrix equation for all angles for the x coordinate is as shown:

$$\begin{bmatrix} t_0^5 & t_0^4 & t_0^3 & t_0^2 & t_0 & 1 \\ 5t_0^4 & 4t_0^3 & 3t_0^2 & 2t_0 & 1 & 0 \\ 20t_0^3 & 12t_0^2 & 6t_0 & 2 & 0 & 0 \\ t_f^5 & t_f^4 & t_f^3 & t_f^2 & t_f & 1 \\ 5t_f^4 & 4t_f^3 & 3t_f^2 & 2t_f & 1 & 0 \\ 20t_f^3 & 12t_f^2 & 6t_f & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_0 \\ x'_0 \\ x''_0 \\ x_f \\ x'_f \\ x''_f \end{bmatrix} \tag{4.73}$$

a_i,b_i,c_i, d_i, e_i and$f_i$are the function parameters for the quintic equation of the i^th joint. x_i^j,$x_i^{j'}$ and$x_i^{j''}$ are the x coordinate value, x coordinate velocity and x coordinate acceleration of the j^th joint angle at time instance t_i In the shorthand matrix form, the quintic equations can be represented as:

$$Ax = B_x$$

It is solved by taking the inverse of the time matrix

$$x = A^{-1}B_x \tag{4.74}$$

After x(t) has been calculated, the quintic functions for y(t) and z(t) is computed using the following:

$$y(t) = y_{T_1} + \frac{(y_{T_2} - y_{T_1})}{(x_{T_2} - x_{T_1})}(x(t) - x_{T_1}) \tag{4.75}$$

$$z(t) = z_{T_1} + \frac{(z_{T_2} - z_{T_1})}{(x_{T_2} - x_{T_1})}(x(t) - x_{T_1}) \tag{4.76}$$

Once the time equations for the three cartesian coordinates x, y and z are calculated using their function parameters, the robotic arm can be moved according to this trajectory. The corresponding time instance values are fed into these time functions every iteration and for the corresponding target coordinates, the Inverse Kinematics solution is calculated. The motors of the arms are then rotated to go to the angles computed in the solution. This process is carried out continuously till the motion is complete. While calculating the Inverse Kinematics solution using the Jacobian Pseudoinverse method, successive configurations must not differ much in terms of difference to ensure smooth motion. This difference or proximity between angles is calculating using a simple root mean square error formula. The proximity of successive angles is ensured by feeding the previous trajectory configuration as the initial configuration to the Inverse Kinematics Jacobian Pseudoinverse routine. Thus, the routine computes the next configuration by taking into account small changes to the initial angles which gives rise to a solution which is very close to the previous configuration.

4.4 MOTOR CONTROL

4.4.1 I2C Protocol

The I2C bus physically consists of 2 active wires and a ground connection. The active wires, called SDA and SCL, are both bi-directional. SDA is the Serial Data line, and SCL is the Serial Clock line. Every device hooked up to the bus has its own unique address, no matter whether it is an MCU, LCD driver, memory, or ASIC. Each of these chips can act as a receiver and/or transmitter, depending on the functionality. The I2C bus is a multi-master bus. This means that more than one IC capable of initiating a data transfer can be connected to it. The I2C protocol specification states that the IC that initiates a data transfer on the bus is considered the Bus Master. Consequently, at that time, all the other ICs are regarded to be Bus Slaves. The address and the data bytes are sent most significant bit first. The start bit is indicated by a high-to-low transition of SDA with SCL high; the stop bit is indicated by a low-to-high transition of SDA with SCL high. All other transitions of SDA take place with SCL low.

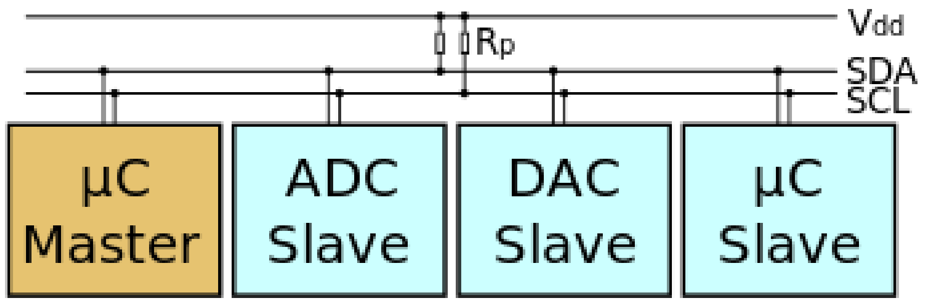


Figure 4.5: Typical I2C bus design

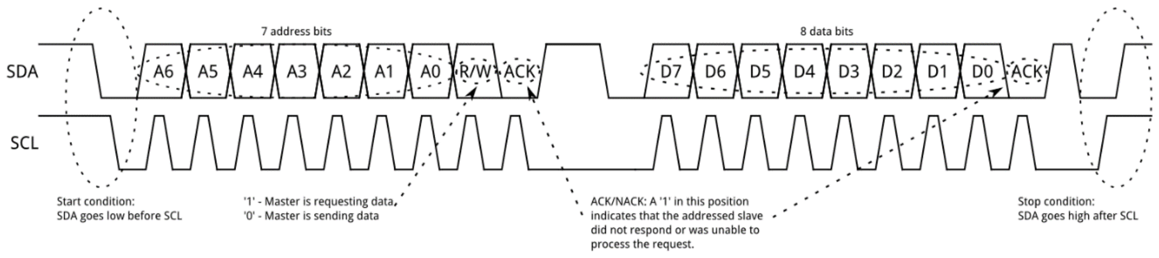

7 address bits
8 data bits
SDA
SCL
A6
A5
A4
A3
A2
A1
A0
R/W
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
Start condition:
SDA goes low before SCL
'1' - Master is requesting data,
'0' - Master is sending data
ACK/NACK: A '1' in this position indicates that the addressed slave did not respond or was unable to process the request.
Stop condition:
SDA goes high after SCL

Figure 4.6: Typical I2C timing

4.4.2 Implementation using PCA9685

The PCA9685 allows communication of PWM values using I2C protocol. The required data is stored in the internal registers and can be accessed for further processing. We have written code in order to access the current motor angle value in order to perform smooth motion of various joints. Each internal register is addressed by an internal address and the value of the internal address is sent along with the data. First, the address is sent in the data packet along with the PWM value in the next data packet.

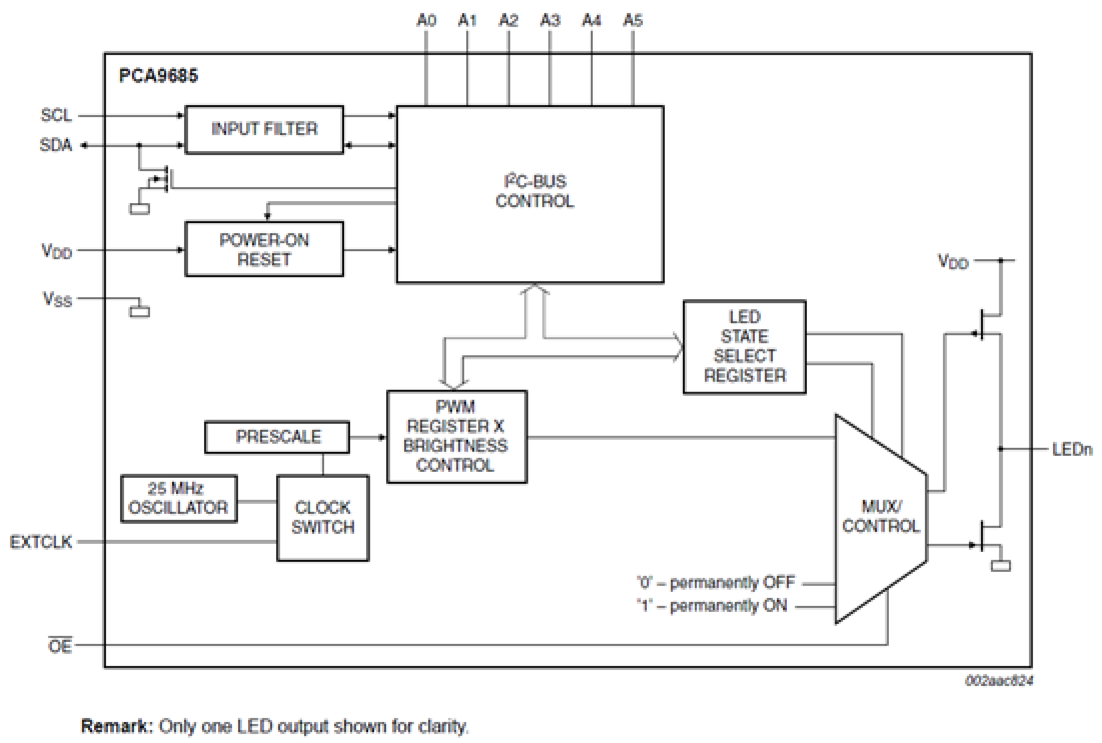


Figure 4.7: Block diagram of PCA9685

4.5 STEREO VISION

4.5.1 Calculating Distance to a Point

The figure shows the diagram of a simplified stereo vision setup, where both cameras are mounted perfectly parallel to each other, and have the exact same focal length.

b is the baseline, or distance between the two cameras.

F is the focal length of a camera.

X_a is the X-axis of a camera.

Z_a is the optical axis of a camera.

P is a real-world point defined by the coordinates X, Y, and Z.

u_L is the projection of the real-world point P in an image acquired by the left camera.

u_R is the projection of the real-world point P in an image acquired by the right camera.

Since the two cameras are separated by distance, both cameras view the same real-world point in a different location on the two-dimensional images acquired. The X-coordinates of points u_L and u_R are given by:

$$u_L = \frac{f.x}{z} \tag{4.77}$$
$$u_R = \frac{f.(x-b)}{z} \tag{4.78}$$

Distance between the two projected points is known as "disparity" and this value is used to calculate depth information, which is the distance between real-world point P and the stereo vision system.

$$disparity = u_L - u_R = \frac{f.b}{z} \tag{4.79}$$
$$depth = \frac{f.b}{disparity} \tag{4.80}$$

The ideal assumptions made earlier for the simplified stereovision system cannot be made for real-world stereovision applications. Even the best cameras and lenses will introduce distortion to the image acquired, Hence to compensate, a typical stereovision system also requires calibration.

The calibration process has calibration target - for example, a grid of dots or a checker-board - and acquiring images at different angles to calculate image distortion, as well as the exact spatial relationship between the two cameras. Stereo camera calibration is used to determine the intrinsic parameters and relative location of cameras in a stereo pair, this information is used for stereo rectification and 3-D reconstruction.

An actual stereovision setup is more complex and would look more like the typical system shown in the figure, but all of the same fundamental principles still apply.

4.5.2 Implementation

The stereo vision algorithm is implemented using a DUO MLX Stereo Camera set up with the following parameters:

Figure 4.8 shows a disparity map calculated using the left and right images.

Parameter	Value
Width	368
Height	224
FPS	20
Scale	None
Mode	Stereo Block Matching
Number of Disparities	2
SAD Window Size	10
Pre-filter Cap	63
Uniqueness Ratio	1
Speckle Window Size	256
Speckle Range	32

Table 4.1: DUO MLX Camera Parameters used

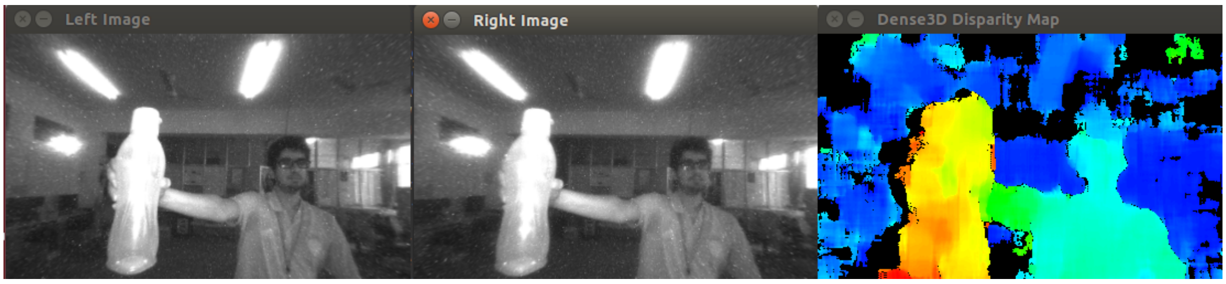


Figure 4.8: Disparity map obtained given a stereo pair

4.6 KALMAN FILTER

The Kalman filter algorithm is used to estimate the most accurate value of the required position(in 3D space), obtained by the stereo camera. Since this value is susceptible to uncertainty and noise, the filter provides a more constant value which increases the stability of the system. The application of the algorithm in stereo vision is highlighted in [30].

The Kalman gain is the relative weight given to the measurements and current state estimate, and can be changed to achieve particular performance.

With a high gain, the filter places more weight on the most recent measurements, and thus follows them more responsively.

With a low gain, the filter follows the model predictions more closely. At the extremes, a high gain close to one will result in a more jumpy estimated trajectory, while low gain close to zero will smooth out noise but decrease the responsiveness.

Following is the explanation of implementation of kalman filter algorithm:

1. PREDICTION STEP:

   $$\bar{x_t} = A_t * x_{t-1} + B_t * u_t \tag{4.81}$$

   Where, $A_t = 1$, is the state transit model of order 1x1. $x_{t-1} = 100$, is the state matrix at t-1 of order 1x1.
   $B_t = 0$, is control input model of order 1x1.
   $u_t = 0$, is control matrix of order 1x1.
   Hence, $\bar{x_t} = 100$, is expected state matrix of order 1x1.

   $$\bar{P_t} = A_t * P_{t-1} * A_t^T + Q_t \tag{4.82}$$

   Where, P_{t-1}= 0, is State correlation matrix of order 1x1.
   $Q_t = 10^{-3}$, is Estimation noise of order 1x1
   Hence, $\bar{P_t} = 10^{-3}$, is Expected state correlation matrix of order 1x1.

2. UPDATING STEP:

   Compute Kalman gain,

   $$K_t = \bar{P_t} * H_t^T(H_t * \bar{P_t} * H_t^T + R_t)^{-1} \tag{4.83}$$

   Where, H_t= 1, is observation model of order 1x1
   R_t= 1, is Measurement noise of order 1x1
   Hence, K_t is kalman gain of order 1x1

   Input with noise Z_t and compute new state,

   $$x_t = \bar{x_t} + K_t(Z_t - H_t * x_t) \tag{4.84}$$

   Where, $Z_(t)$ is Observation matrix of order 1x1. Its values are taken in random.
   x_t is state matrix of order 1x1

   Update covariance,

   $$P_t = (I - K_t * H_t)\bar{P_t} \tag{4.85}$$

   Where, P_t is State correlation matrix of order 1x1

Return updated x_t,P_t
Repeat the following steps till the differences between the successive readings is 0.1.

4.7 OBJECT INTERACTION

The ability to interact with objects is very important for an intelligent robot. For such a robot to be useful to industries or consumers, it must have an understanding of its surroundings and must be able to physically manipulate such objects. Picking up and dropping of objects is one of the major aspects of ARDOP. It involves the integration of a multitude of different areas in robotics. The overall block diagram and functional block diagram of the process is described by Figure 4.9 and Figure 4.10.

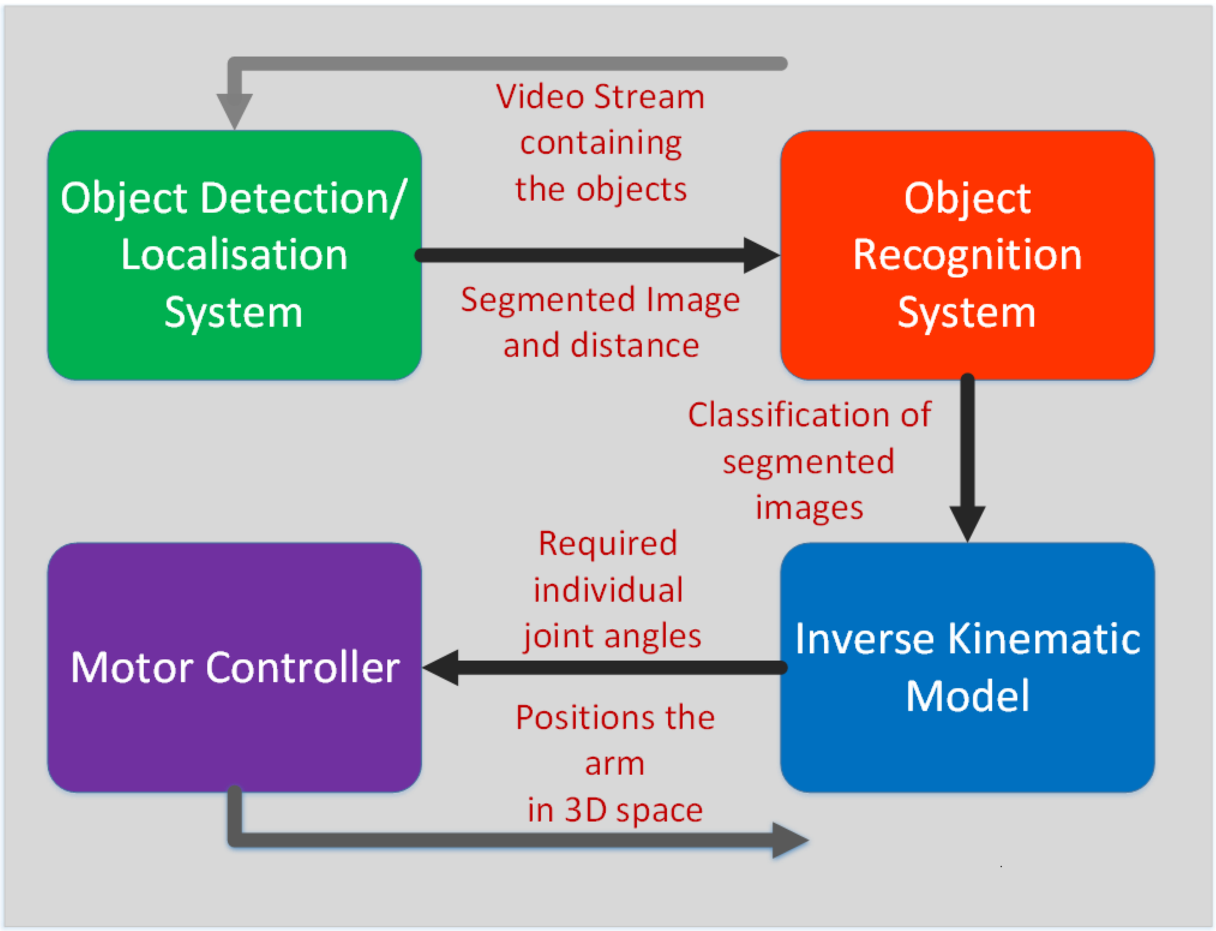

Video Stream containing the objects
Object Detection/ Localisation System
Object Recognition System
Segmented Image and distance
Classification of segmented images
Required individual joint angles
Motor Controller
Inverse Kinematic Model
Positions the arm in 3D space

Figure 4.9: Overall block diagram

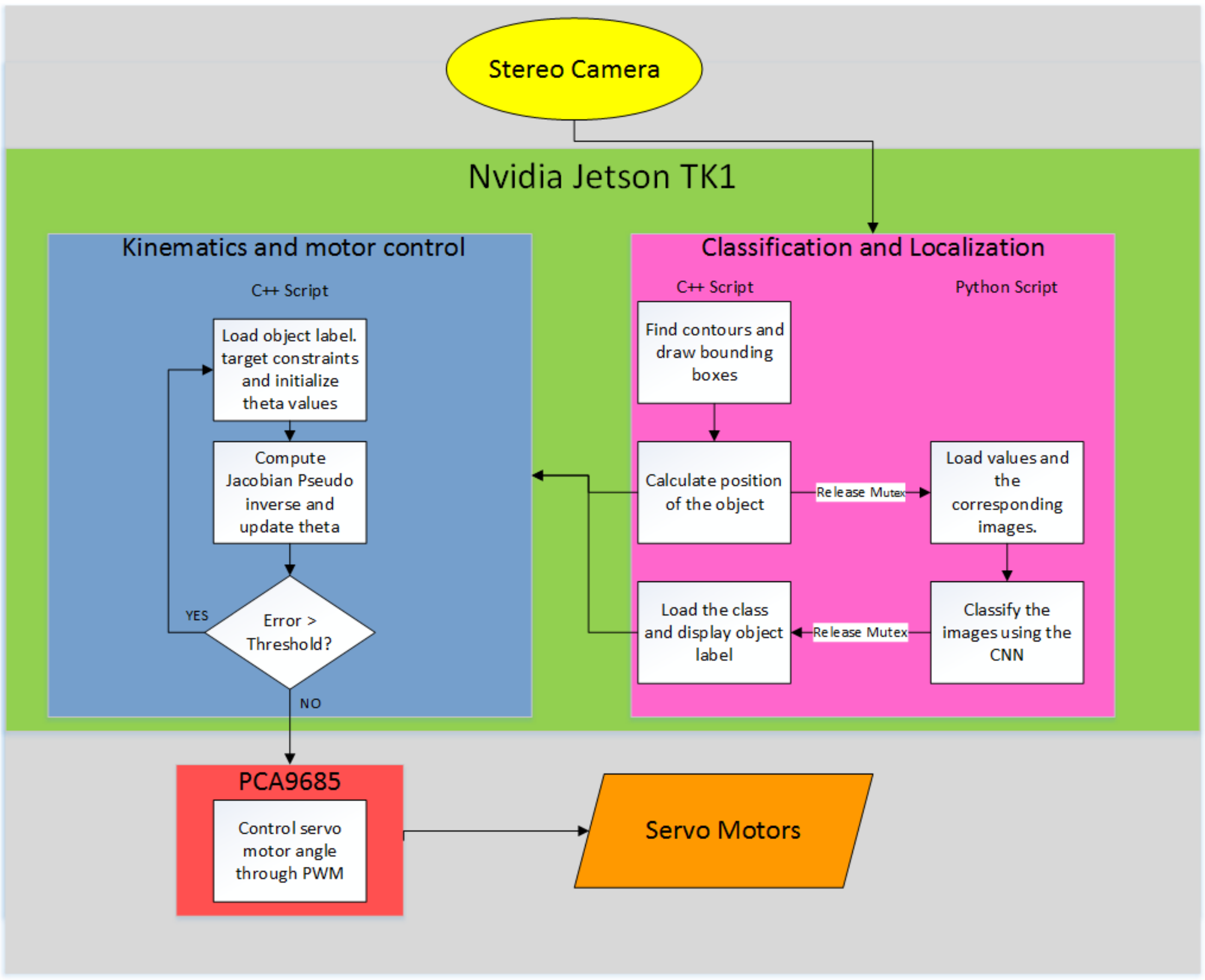

Stereo Camera
Nvidia Jetson TK1
Kinematics and motor control
C++ Script
Load object label. target constraints and initialize theta values
Compute Jacobian Pseudo inverse and update theta
YES
Error > Threshold?
NO
Classification and Localization
C++ Script
Python Script
Find contours and draw bounding boxes
Calculate position of the object
Release Mutex
Load values and the corresponding images.
Classify the images using the CNN
Release Mutex
Load the class and display object label
PCA9685
Control servo motor angle through PWM
Servo Motors

Figure 4.10: Functional block diagram

The various aspects and steps involved in picking up of objects are listed below.

1. Object localization and classification

   The objects in the scene must first be localized and classified. This is necessary to give the robot enough information about the objects and its position in the camera frame. As Convolution Neural Networks (CNN's) are used for object classification, the exact regions of the frame from the camera stream that contain the objects must be used to classify. A contour based localization system is used for this purpose, as seen in [31]. A Canny edge detection [32] serves as a pre-processing step in order to obtain better results. Bounding boxes are constructed over the contour regions that correspond to objects in the scene and the images obtained from these boxes are fed to the CNN for classification. After this procedure, the identities of the objects as well as its position with respect to the image are known.

2. Distance computation

   After the objects in the scene have been classified and localized, their exact positions in 3-D space must be computed for the robot to be able to pick it up. The Stereo Camera is used to obtain the real-world coordinates of the objects in the frame with respect to the camera. The values obtained from the camera are mapped properly to ensure that the distances are with respect to a non-skewed camera plane. After this has been computed, the coordinates are translated to a new origin which is the origin of the kinematic model. These distances can now be used directly as the target value of the end effector to compute the Inverse Kinematics solution.

3. Inverse kinematic solution and trajectory planning

   After the coordinates of the object to be picked up have been computed, the Inverse Kinematics solution is computed taking the coordinates as the target position. The Jacobian Pseudoinverse iterative solution is used to find this solution. If there are multiple objects in the scene, it is checked if any of the arm configurations intersect with the positions of the other objects in the scene. If it does, then the other object is moved away so that its position does not overlap with any of the configurations during the pickup motion. Appropriate trajectory planning methods are used to accomplish this task and ensure smooth object pickup.

4. Final joint pose computation

   For the robot arm gripper to be able to pick up the object, the final angle in the arm that rotates the gripper must be oriented in such a way that the gripper is parallel to the ground or the object. This means that the gripping motion must always be sideways to enable perfect object interaction. There is not a direct solution to this problem as the computed joint angles are not always constant, even for the same target position. The pose angle must thus be calculated as a function of the other four joint variables to ensure that the gripper is always parallel to the ground no matter what the other joint variable values are.

   If the gripper is parallel, it means that the x coordinate of the final joint variable kinematic frame must be zero or minimized. This means that the a_x component of the Forward Kinematics matrix must be minimized.

From the Forward Kinematics matrix,

$$\begin{aligned} a_x = cos\theta_1 cos\theta_2 cos\theta_3 cos\theta_4 sin\theta_5 - cos\theta_1 cos\theta_2 sin\theta_3 cos\theta_5 + \\ cos\theta_1 sin\theta_2 sin\theta_4 sin\theta_5 + sin\theta_1 sin\theta_3 cos\theta_4 sin\theta_5 + sin\theta_1 cos\theta_3 cos\theta_5 \end{aligned} \tag{4.86}$$

Minimizing,

$$\frac{\partial a_x}{\partial \theta_5} = 0 \tag{4.87}$$

After solving,

$$\theta_5 = n\pi - tan^{-1}(\frac{A}{B}) \; \forall \; n \in \mathbb{Z} \tag{4.88}$$

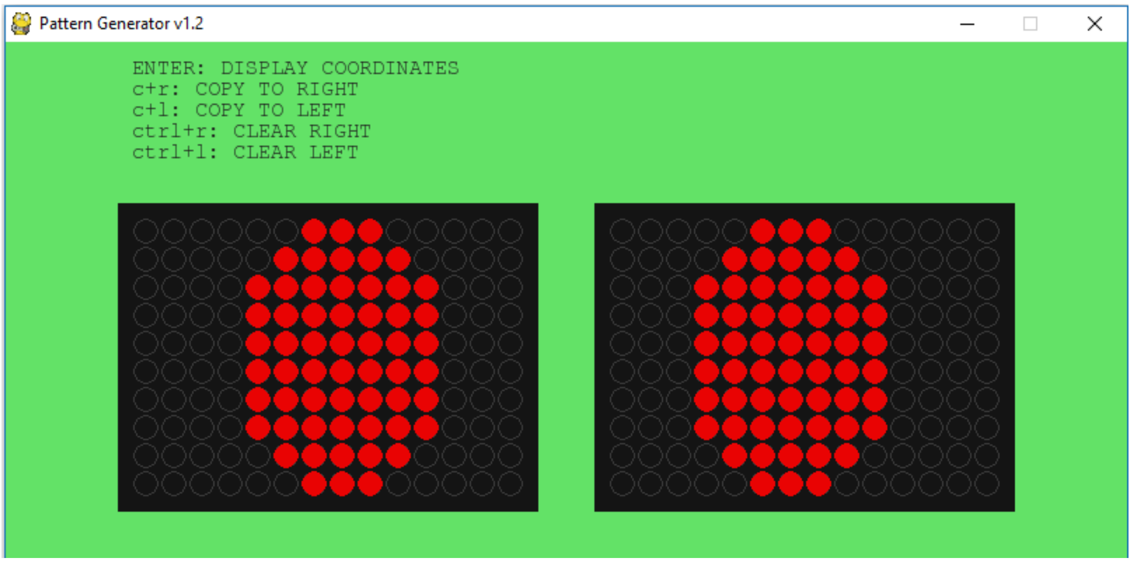
Where,

$$A = cos\theta_1 cos\theta_2 cos\theta_3 cos\theta_4 sin\theta_5 + cos\theta_1 sin\theta_2 sin\theta_4 sin\theta_5 + sin\theta_1 sin\theta_3 cos\theta_4 sin\theta_5 \tag{4.89}$$

$$B = cos\theta_1 cos\theta_2 sin\theta_3 cos\theta_5 - sin\theta_1 cos\theta_3 cos\theta_5 \tag{4.90}$$

4.8 ROBOTIC EMOTION DISPLAY

The required patterns on the LED matrix is generated using a python script. The corresponding code is generated and this code can be used in the Arduino code. The python script greatly reduced the time and complexity in converting designed patters into the corresponding code. Figure 4.11 represents an example pattern generated by the python script.



```
ENTER: DISPLAY COORDINATES
c+r: COPY TO RIGHT
c+l: COPY TO LEFT
ctrl+r: CLEAR RIGHT
ctrl+l: CLEAR LEFT
```

Figure 4.11: an example pattern generated by the python script.

Chapter 5

RESULTS

5.1 OBJECT RECOGNITION

The model was trained for 300 iterations and the final test accuracy was found to be 100%. The learning curve is plotted in Figure 5.1.

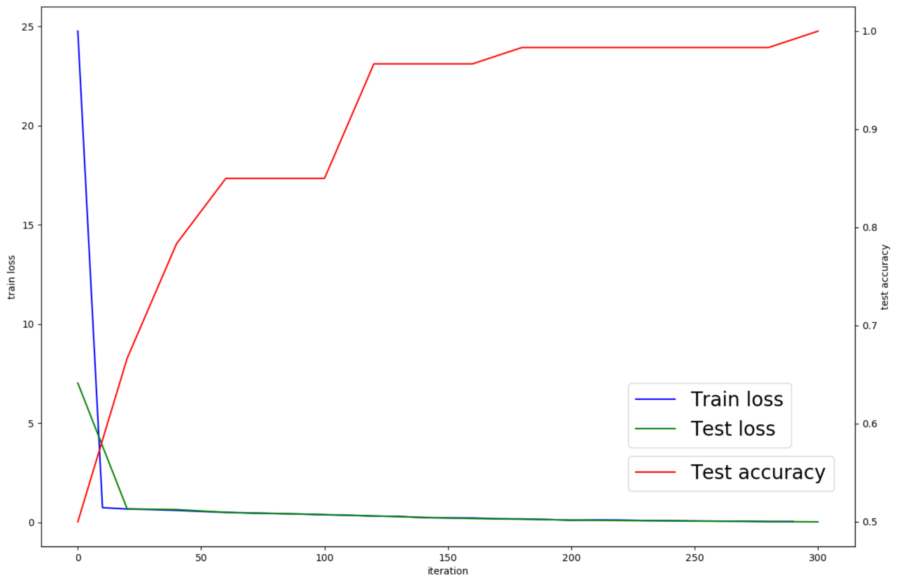


Figure 5.1: Learning curve observed while training the dataset

The model also perform excellently while classifying the objects during the object interaction process. The frames from the camera stream is segmented and the classified output is shown in Figure 5.2.

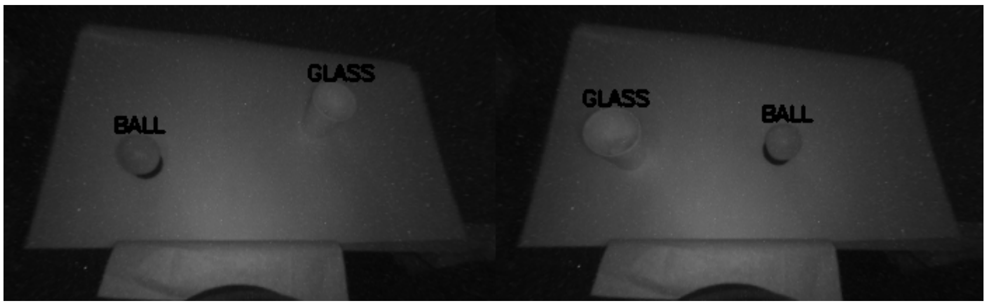


Figure 5.2: Classified objects in two camera frames

5.2 KINEMATICS

5.2.1 Analytical solution

The analytical equations obtained have been verified to produce the accurate angle solutions in the workspace of each arm. This approach is not robust as it requires the knowledge of the orientation matrix of the end effector. It becomes impossible to compute the Euler rotation matrix for any given position without the knowledge of the joint angles. Hence, this approach is not practical even though it provides multiple solutions for our joint angles.

5.2.2 Iterative methods

The joint values can be computed without the knowledge of the orientation of the end effector using this approach. We can only obtain one solution for the angles but this does not affect our result as it ensures that our end effector reaches the target position without considering the final orientation. Any constraints can be accommodated by suitably modifying the iterative function. The results of these methods have been simulated in MATLAB and verified to produce accurate inverse kinematic results.

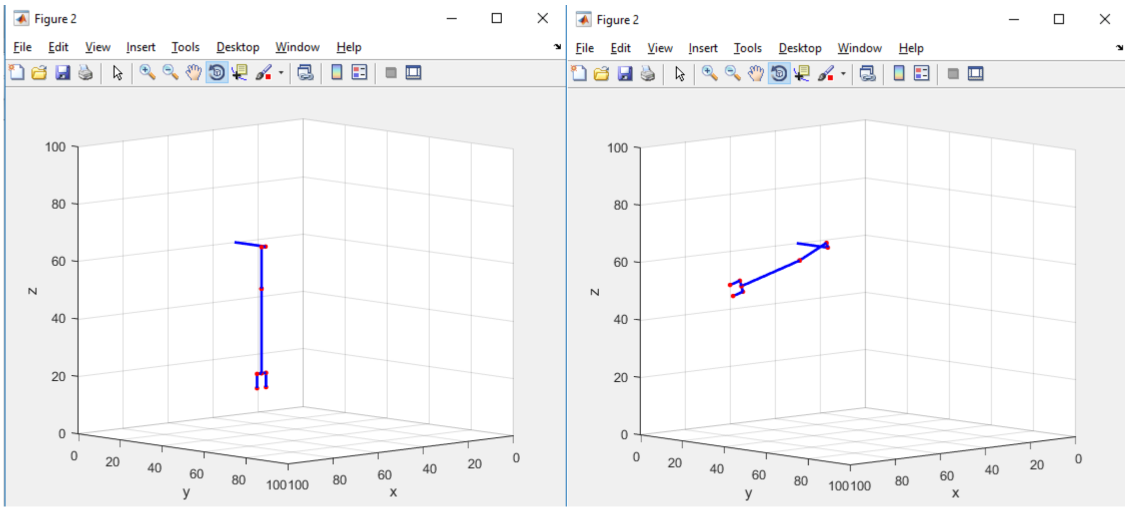


Figure 2
File Edit View Insert Tools Desktop Window Help
z
x
y
Figure 2
File Edit View Insert Tools Desktop Window Help
z
x
y

Figure 5.3: Simulation of inverse kinematics solutions for various end effector positions.

5.3 PATH PLANNING

A complete MATLAB UI interface has been created using Guide. On inputting the x, y and z coordinates of the target, the application is able to simulate the motion of the robotic arm using the implemented path planning techniques. The robot arm has the same dimensions, characteristics and angle constraints of the actual arm of ARDOP and the simulation is thus a genuine estimator of the behavior of various kinematic and motion algorithms.

When linear path planning is implemented, the following graph was obtained after plotting the joint angle time function for each of the joints.

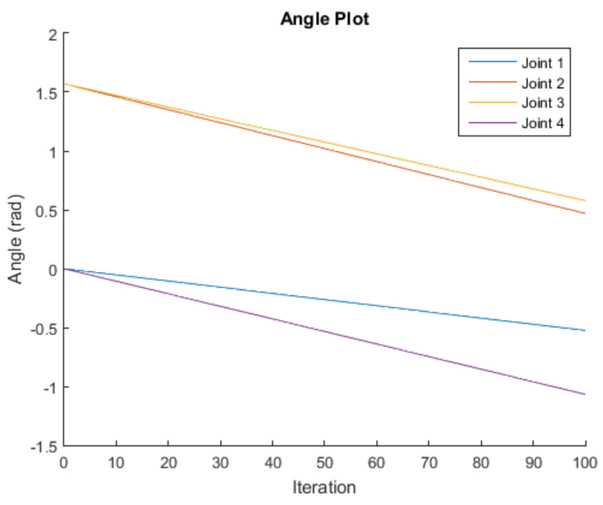


Figure 5.4: Linear path angle plot

As seen in Figure 5.4, the variation of the joint angles are linear and vary from the start to the end configuration.

For a cubic path, the following plots were obtained for the variation of joint angles and their velocities:

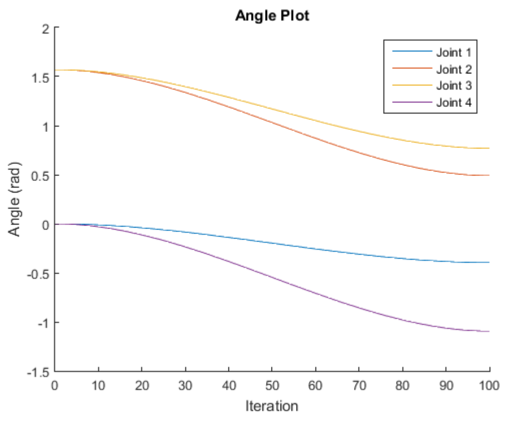


Figure 5.5: Variation of joint velocities in cubic path,

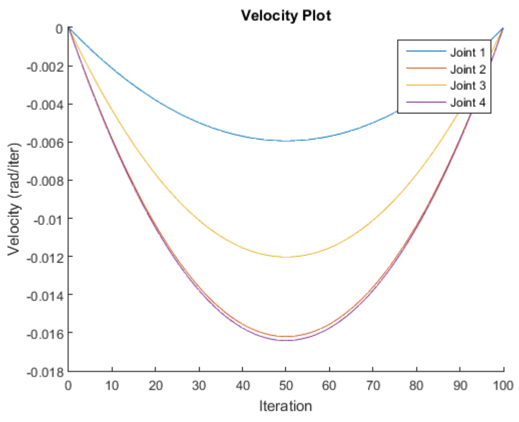


Figure 5.6: Variation of joint angles in cubic path

The changes in velocity and joint angles are nonlinear as expected. The end constraints of the angles and velocities are satisfied as shown in the figures Figure 5.5 and Figure 5.6.

For a piecewise path, with a constant velocity phase, the following plots were obtained:

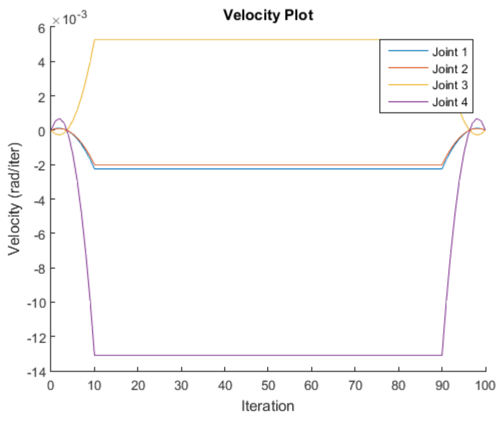


Figure 5.7: Variation of joint velocities in piecewise path,

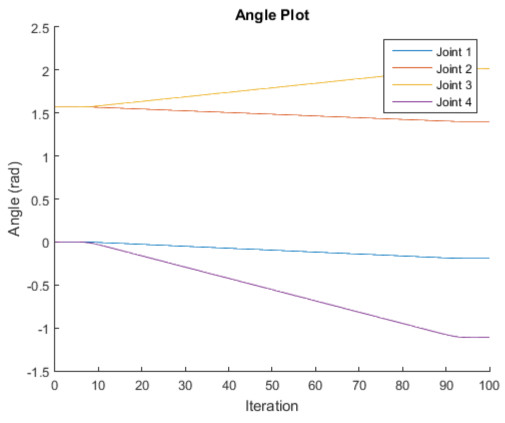


Figure 5.8: Variation of joint angles in piecewise path

As expected from the model, the velocities change gradually till the middle phase and remains constant for the majority of the motion. This is seen in the above 5.7 and5.8 After the constant velocity phase, the velocities again gradually decrease to zero.

The MATLAB application plots the end-effector trajectory while carrying out Cartesian Path Planning as this kind of planning takes into account the path of the end-effector. This helps in debugging the computed path.

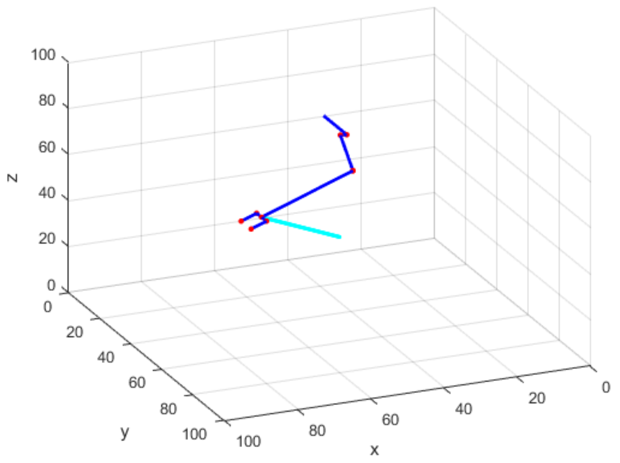


Figure 5.9: End effector trajectory

The Figure 5.9 shows the moved arm along with the trajectory taken by the end-effector.

The straight-line cartesian path using linear equations give the following plots for variance of the target coordinate with respect to time:

As seen in the 5.10, the variation of the coordinates is linear as expected. Hence, they trace a linear path from the initial end-effector target to the final target coordinate.

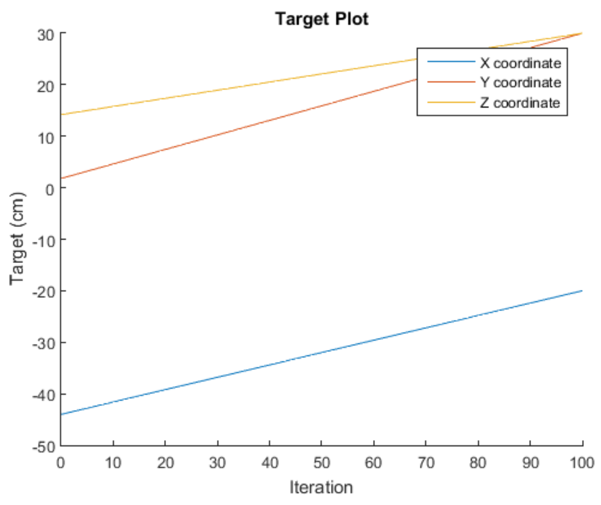


Figure 5.10: Variation of end-effector coordinates in linear cartesian path

The straight-line quintic model gives the following results for end-effector position, velocity and acceleration:

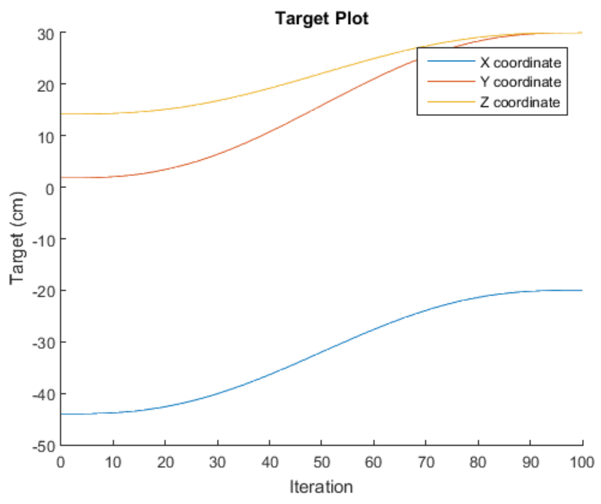


Figure 5.11: Variation of end-effector coordinates in linear quintic path

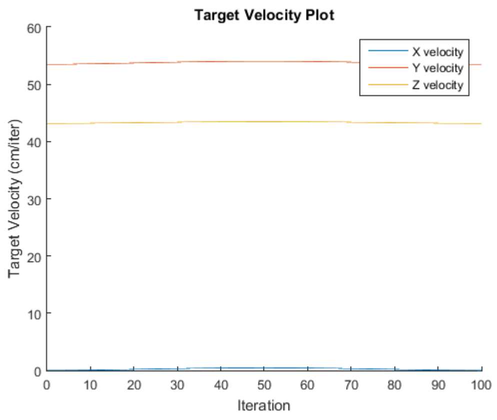


Figure 5.12: Variation of end-effector velocities in linear quintic path

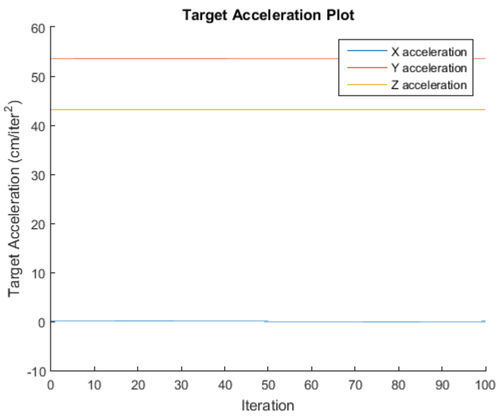


Figure 5.13: Variation of end-effector accelerations in linear quintic path

As expected, the velocity and acceleration constraints given to the x coordinate are satisfied as seen in the 5.12 and 5.13. Although the x, y and z coordinates have nonlinear time functions as seen in the figure, their combined variation, results in a linear motion of the end-effector target which satisfies all required conditions.

Hence all the path planning models give correct and expected results when simulated. All the give constraints are satisfied by the generated paths. Clear and elegant trajectories are obtained provided the Inverse Kinematic Solutions at the required points are possible. As long as the motion of the end-effector is kept within the workspace of the robotic arm, valid and precise path planning solutions are generated.

5.4 OBJECT INTERACTION

The process mentioned above was successfully implemented and a success rate of 90% observed over 30 attempts to pick up the ball. The processes are seen in Figure 5.14. The class of the objects in the frame as well as the position of each of them are obtained. The the inverse kinematic solution was then successfully obtained, thereby positioning the end effector at the required point, before picking up the object.

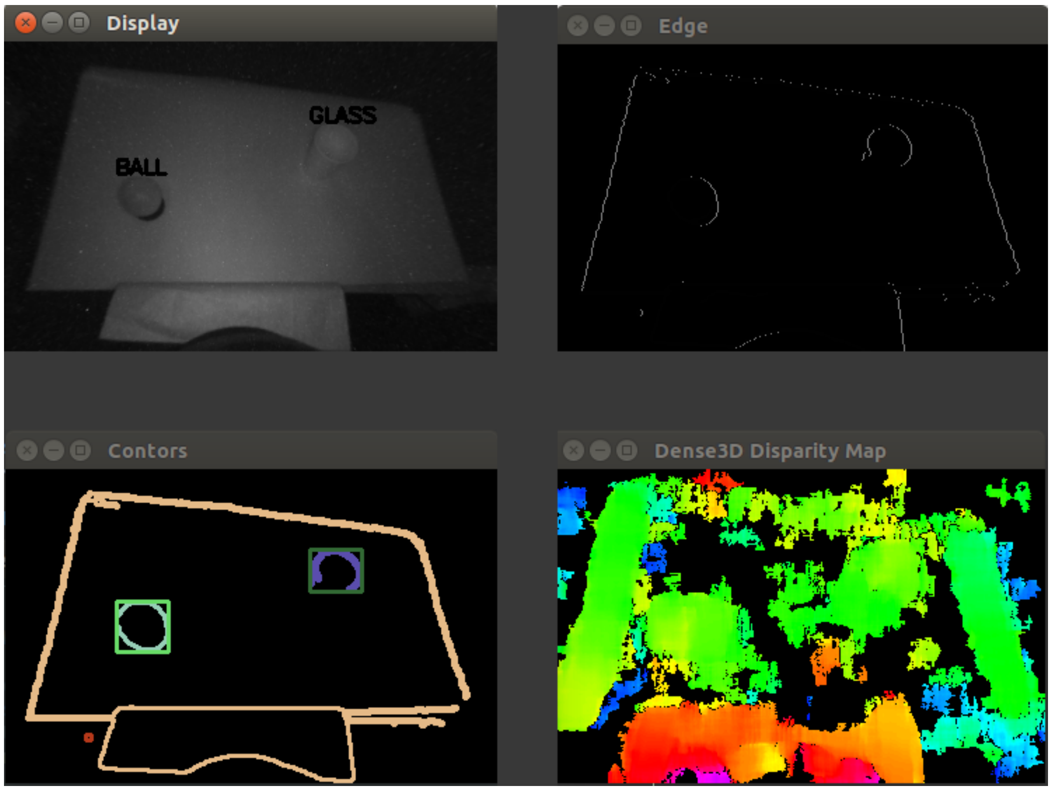


Figure 5.14: Sequence of operations carried out to localize and classify objects in the frame

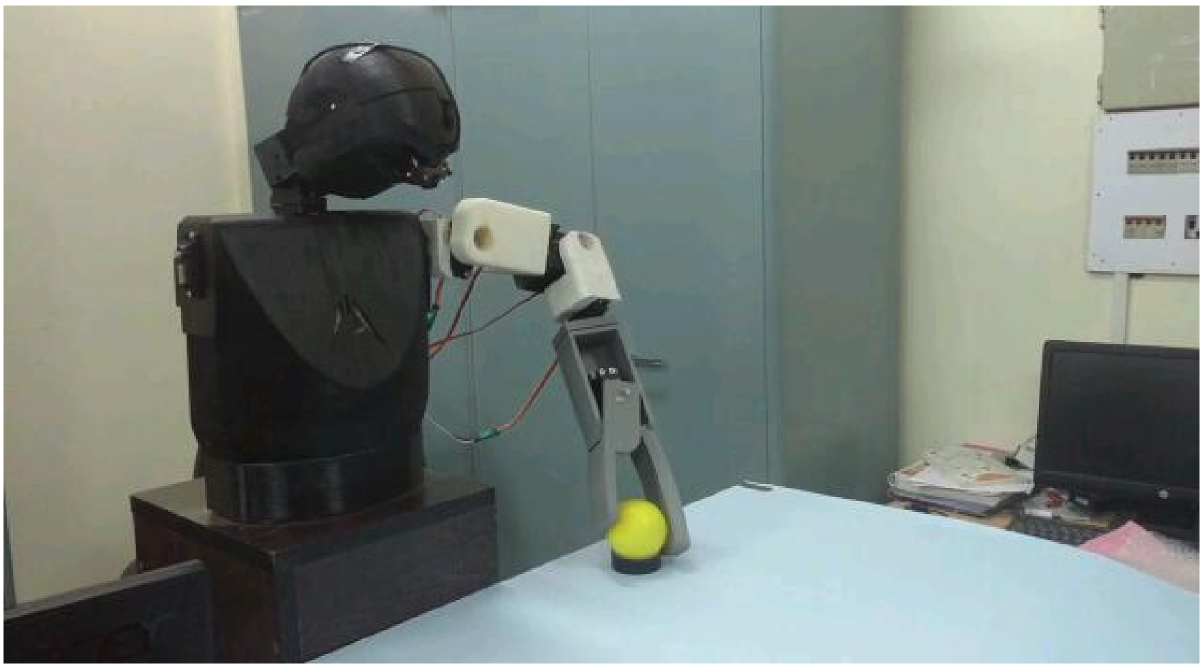


Figure 5.15: ARDOP picking up a ball

Chapter 6

CONCLUSION

The upper torso of the humanoid robot has been 3D printed and assembled. All the motors have been fixed and interfaced with the NVIDIA Jetson TK1 processor using the I2C protocol. The motors are interfaced through the PCA9685 I2C driver. The processor is programmed to send appropriate signals to the PCA to enable the motors to be rotated at particular angles with a high degree of accuracy.

An emotion display system for the robot has been developed which would help make the robot more user friendly and interactive. An LED array architecture using shift registers is used for this purpose. A python script has been developed so that the required pattern can be displayed on the LED array. The LED display pattern depends on the set of coordinates set on the microcontroller. The python script can be used to produce these coordinates for any arbitrary pattern which can then be set for that emotion to be displayed.

An object recognition model has been implemented using a convolutional neural network. Training data is produced for a number of images using and the model is trained, resulting in a highly accurate, real-time classification system. The algorithm has been implemented on the Jetson TK1 using the caffe library. The depth to the recognized objects were obtained and the information was relayed to the kinematic system.

The kinematics system computes required joint angles for the gripper to move to the target position. Path planning algorithms are used to compute joint and gripper trajectories for the arm to interact with the object. The grippers are orientated appropriately to enable effective object interaction.

More sophisticated algorithms can be used for object classification and localization. Better and more precise joints can be made to enable the robot to perform more intricate manipulative tasks. The platform can be made more generic such that they can be ported to other intelligent systems. A distributive platform for machine learning can be developed which would help developers from around the world to share their learning models with each other.

References

[1] John Illingworth and Josef Kittler. A survey of the hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988.

[2] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Frank L Lewis, Chaouki T Abdallah, and Darren M Dawson. *Control of robot manipulators*, volume 236. Macmillan New York, 1993.

[7] John Q Gan, Eimei Oyama, Eric M Rosales, and Huosheng Hu. A complete analytical solution to the inverse kinematics of the pioneer 2 robotic arm. *Robotica*, 23(01):123–129, 2005.

[8] Kevin Cleary and Thurston Brooks. Kinematic analysis of a novel 6-dof parallel manipulator. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 708–713. IEEE, 1993.

[9] Jingguo Wang, Yangmin Li, and Xinhua Zhao. Inverse kinematics and control of a 7-dof redundant manipulator based on the closed-loop algorithm. *International Journal of Advanced Robotic Systems*, 7(4):1–9, 2010.

[10] Yoshihiko Nakamura and Hideo Hanafusa. Inverse kinematic solutions with singularity robustness for robot manipulator control. *ASME, Transactions, Journal of Dynamic Systems, Measurement, and Control*, 108:163–171, 1986.

[11] Chris Welman. *Inverse kinematics and geometric constraints for articulated figure manipulation.* Simon Fraser University, 1994.

[12] Andreas Aristidou and Joan Lasenby. *Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver.* University of Cambridge, Department of Engineering, 2009.

[13] L-CT Wang and Chih-Cheng Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 7(4):489–499, 1991.

[14] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17(1-19):16, 2004.

[15] Johnson YS Luh and Chyuan S Lin. Optimum path planning for mechanical manipulators. *Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control*, pages 142–151, 1981.

[16] R.N. Jazar. *Theory of Applied Robotics: Kinematics, Dynamics, and Control.* Springer, 2010.

[17] A Gasparetto and V Zanotto. A new method for smooth trajectory planning of robot manipulators. *Mechanism and machine theory*, 42(4):455–471, 2007.

[18] Sonja Macfarlane and Elizabeth A Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on Robotics and Automation*, 19(1):42–52, 2003.

[19] D Costantinescu and EA Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *Journal of robotic systems*, 17(5):233–249, 2000.

[20] Wenfu Xu, Cheng Li, Bin Liang, Yu Liu, and Yangsheng Xu. The cartesian path planning of free-floating space robot using particle swarm optimization. *International Journal of Advanced Robotic Systems*, 5(3):27, 2008.

[21] David Marr and Tomaso Poggio. A computational theory of human stereo vision. *Proceedings of the Royal Society of London B: Biological Sciences*, 204(1156):301–328, 1979.

[22] Don Murray and James J Little. Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*, 8(2):161–171, 2000.

[23] Danica Kragic, Mårten Björkman, Henrik I Christensen, and Jan-Olof Eklundh. Vision for robotic object manipulation in domestic settings. *Robotics and autonomous Systems*, 52(1):85–100, 2005.

[24] Miguel Angel Garcia and Agusti Solanas. 3d simultaneous localization and modeling from stereo vision. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 847–853. IEEE, 2004.

[25] Greg Welch and Gary Bishop. An introduction to the kalman filter. 1995.

[26] Andrew C Harvey. *Forecasting, structural time series models and the Kalman filter*. Cambridge university press, 1990.

[27] Motilal Agrawal and Kurt Konolige. Real-time localization in outdoor environments using stereo vision and inexpensive gps. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 1063–1068. IEEE, 2006.

[28] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[29] De Xu, Carlos A Acosta Calderon, John Q Gan, Huosheng Hu, and Min Tan. An analysis of the inverse kinematics for a 5-dof manipulator. *International Journal of Automation and Computing*, 2(2):114–124, 2005.

[30] David J Kriegman, Ernst Triendl, and Thomas O Binford. Stereo vision and navigation in buildings for mobile robots. *IEEE Transactions on Robotics and Automation*, 5(6):792–803, 1989.

[31] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.

[32] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.