Autonomous Motion Control of Industrial Manipulator

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech

in

Mechanical Engineering

BY

Avu Kushal Shrinivas	**(111115016)**
Ashwath Balaje	**(111115025)**
Pratik J	**(111115073)**

Dept. of Mechanical Engineering
National Institute of Technology
Tiruchirapalli - 620015

May 2019

Autonomous Motion Control of Industrial Manipulator

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech

in

Mechanical Engineering

BY

Avu Kushal Shrinivas	**(111115016)**
Ashwath Balaje	**(111115025)**
Pratik J	**(111115073)**

Dept. of Mechanical Engineering
National Institute of Technology
Tiruchirapalli - 620015

May 2019

Bonafide Certificate

This is to certify that the project titled **Autonomous Motion Control of Industrial Manipulator** is a bonafide record of the work done by

Avu Kushal Shrinivas **(111115016)**
Ashwath Balaje **(111115025)**
Pratik J **(111115073)**

in partial fulfillment of the requirements for the award of the degree **Bachelor of Technology** in **Mechanical Engineering** at the **NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI,** during the year 2018-2019.

Dr. K Pannirselvam **Dr. S P Sivapirakasam**

Project Guide Head of Department

Project Viva-voce held on:

Internal Examiner **External Examiner**

Abstract

Industrial Manipulators are widely used devices in the manufacturing industry today. They are used to replace repetitive human effort on assembly lines, where they are capable of producing a constant working efficiency regardless of the number of iterations the same task is performed. In an automated assembly line, these manipulators perform a wide range of tasks from picking and placing objects to assembling parts using fasteners or even automated welded joints. This project harnesses the versatility of these manipulators and explores their utility in outdoor environments in a fully automated operation mode. The prime application considered here is vehicle tire inflation at petrol bunks or automated car wash facilities.

In this thesis we test the capability of a manipulator along with an assisting vision system, to perform the task of tire inflation. This thesis talks about the manipulator motion control algorithms and the vision systems it would require to perform such a task in a dynamic, unpredictable outdoor environment as opposed to the controlled factory environment that these manipulators are currently used in.

Keywords: Industrial manipulator, vision system, motion control algorithms, automated tire inflation application

ACKNOWLEDGEMENTS

We are grateful to Dr. K Pannirselvam, our project guide for his constant support and encouragement right from the beginning. We thank him for giving us the freedom to explore different methodologies and for his support despite multiple failed attempts initially. Without his inputs this project would not have reached completion within the specified deadline.

We would also like to thank Mr. Rhenius, the Robotics Lab incharge at the Siemen's Center of Excellence at the National Institute of Technology, Tiruchirappalli for providing us with technical support while we worked on the KUKA KR-16 robot.

Contents

Abstract iii

Acknowledgements iv

Table of Contents iv

List of Tables vii

List of Figures viii

List of Symbols and Abbreviations x

1 Introduction 1

1.1 General 1

1.2 Problem Description 2

1.3 Objectives 2

2 Literature Review 3

2.1 Vision Systems 3

2.2 Motion Control 4

2.2.1 Forward Kinematics 5

2.2.2 Inverse Kinematics 5

2.2.3 Trajectory Generation 6

2.3 Industrial Manipulator 7

3 Methodology and Systems Design 10

3.1 Vision System 10

3.1.1 R-CNN 11

3.1.2 Valve Detection 14
3.2 Motion Algorithm 19
3.3 Local Network Establishment 20

4 Experimentation and Results **23**
4.1 Experimental Setup 23
4.2 Vision System Results 24
4.3 Motion Algorithm Results 26

5 Summary and Conclusions **30**
5.1 Summary 30
5.2 Conclusions 30
5.3 Scope for Future Work 31
5.4 Other Areas of Application 31

References **32**

Appendices **33**

A MATLAB CODE- Vision System **34**

B MATLAB Code- Motion Control **41**

List of Tables

2.2 Tested Vision Systems 3

4.1 Experimental Results 26

List of Figures

2.1 2-DoF Manipulator Parameters 4
2.2 Inverse Kinematics Solution for simple 2-DoF manipulator 5
2.3 Simple 6-link manipulator(left) and its schematic(right) 8
2.4 3D work-space of a manipulator 8
2.5 Schematic showing the net work-space and the dexterous work-space . 9
2.6 KUKA KR-16 9

3.1 Stereo Camera Setup 11
3.2 Parking region for filling air at a petrol bunk 12
3.3 R-CNN Architecture 12
3.4 Hidden Layer Architecture 13
3.5 R-CNN Test Results 14
3.6 Test Image 15
3.7 Dilated Gradient Mask with detected tire edges 15
3.8 Filled holes for diameter calculation 16
3.9 SURF Features 16
3.10 Putatively matched points 17
3.11 Detected Box, first iteration 18
3.12 Triangulation Method for Depth Calculation 19
3.13 Network Schematic 21
3.14 X66 Port on the KRC 21
3.15 Experimental Network Setup 22

4.1 Experimental Setup 23
4.2 Schematic of Experimental Setup 24
4.3 Tire position on first run 24

4.4 SURF features for first run 25
4.5 Experimental putative matches for first run 25
4.6 Detected Box for first run 27
4.7 First run result 28
4.8 Second run result 28
4.9 Third run result 29

List of Symbols and Abbreviations

DoF - Degree of Freedom

PDAF - Phase Detection Auto Focus

R-CNN - Region-Convolutional Neural Network

BBox - Bounding Box

SVM - Support Vector Machine

SURF - Speeded Up Robust Features

SIFT - Scale Invariant Feature Transform

FoV - Field of View

KSS - KUKA System Software

KRL - KUKA Robot Language

P2P - Point to Point

KRC - KUKA Robot Controller

KCT - KUKA Control Toolbox

Chapter 1

Introduction

1.1 General

Industrial manipulators have revolutionized manufacturing methodologies by bringing human effort on the assembly line to a bare minimum, while increasing the production efficiency and decreasing the batch time. However, in a factory environment, these manipulators are programmed to do a particular task such as spot weld, at exactly the same spot on every article on the assembly line. Furthermore, the articles are always positioned identically, in the same orientation and at the same location by controlling the conveyor system accordingly. This means the manipulator has no need to make dynamic adjustments each time and its motion control algorithms can be hard-coded or programmed offline using a teach pendant. This project aims to use the versatility of these manipulators and combine it with custom designed vision systems to create a fully automated system. The primary application selected for this project is that of inflating vehicle tires using our automated system.

Tires inflated to the rated optimum pressure, provide the best fuel efficiency. However, often drivers are in too much of a hurry to wait for the air pump operator to arrive and inflate the tires to the optimum pressure. This results in multiple vehicles riding at sub-optimal tire pressures, despite just having visited a petrol bunk. This leads to higher rolling resistance which leads to a cascading chain of events that eventually leads to low fuel efficiency. This also has significant effects on the environment, as we end up using more fuel than required and inevitably cause higher emissions. This project aims

to counter this by developing an automated system for the same

1.2 Problem Description

The goal of our project is to create an automated system using an industrial manipulator to perform tire inflation operation on vehicles at locations such as petrol bunks, automated car wash facilities etc. The problem statement only deals with developing a system to aid in the inflation operation, however the actual inflation operation is out of the scope of this project.
The problem statement is divided into the following parts:

1. Detecting whether a car has been parked in front of the air filling station
2. Calculating the position and orientation of the tire
3. Detecting the valve co-ordinates
4. Generating a trajectory for the end-effector of the manipulator to reach the detected valve co-ordinates

1.3 Objectives

1. Design a fast Neural Network to detect whether a car has been parked in front of the air filling station
2. Design a vision system that calculates both distance and orientation of the car tire
3. Write a fast trajectory planning algorithm for moving the end-effector to the valve

Chapter 2

Literature Review

2.1 Vision Systems

Multiple vision systems (camera and non-camera sensor based) were studied for their merits and demerits (Tab. 2.2), and the most suitable system was selected.

Type	Merits	Demerits
3D Time of flight Sensor (3D ToF)	High accuracy, high speed means it can be used for real time applications	Expensive and complex setup
3D Mobile Scanning (Stitching multiple images)	Doesn't require any specialized equipment, can be done with the help of software applications on a cellular phone	Requires multiple images of the object in order to stitch an image, taking a lot of time.
Kinect System (Skeletal Tracking)	The depth map is constructed by analyzing a speckle pattern of infrared laser light and determining the distance by deformations in the pattern, hence very accurate	Proprietary and expensive software.
Stereo camera setup (Depth Triangulation)	It can be programmed to be more intelligent in making decisions. It also performs the depth calculation and valve detection in a single go	Depth information is not as accurate as ToF sensors

Table 2.2: Tested Vision Systems

Tire inflation is meant to be a highly cost-effective process. Using an expensive vision system will become redundant towards achieving our goal. Taking this primary consideration into account the 3D ToF and Kinect System were eliminated. The 3D Mobile Scanning system is a cheap alternative. Its working involves taking multiple pictures of a stationary object and stitching the images together to produce a 3D scan of the object. However, this process goes against the second main aim of the project, i.e., making the tire inflation process time optimized. The stereo camera setup is a cheap option plus it serves the added benefits of providing ability to program algorithms suited to dynamically changing environments and at the same time it can produce depth calculation and valve orientation data in a single iterative run. This system also works with cheap, low-resolution camera sensors such as those available in bottom-end smartphones eliminating the need for a high-tech camera setup. These advantages outweigh the disadvantage of minute inaccuracies in detection, subject to uncontrollable parameters like dust or shadows. Thereby, the stereo camera setup was chosen as the operating vision system for this project.

2.2 Motion Control

Motion control of a robot manipulator involves basic kinematic relations to traverse efficiently from joint space coordinates to cartesian coordinates. Joint space coordinate system refers to the parameters of a particular link defined wrt. a coordinate system centered at one of its joints. The parameters a_1 and a_2 shown in Fig.2.2 are joint space parameters and x_1, x_2, y_1, y_2 are cartesian parameters.

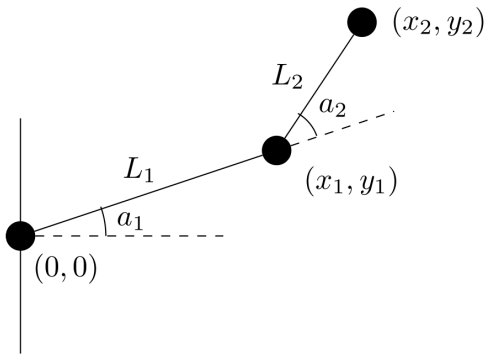


Figure 2.1: 2-DoF Manipulator Parameters

2.2.1 Forward Kinematics

The process of determining the cartesian parameters given the joint space parameters is known forward kinematics. The forward kinematics solution [1] for Fig.2.2 is:

$$x_2 = L_1 \cos a_1 + L_2 \cos a_1 + a_2 y_2 = L_1 \sin a_1 + L_2 \sin a_1 + a_2$$

This process is useful to determine the end-effector location, but is however of no use in determining a trajectory of motion of a robot. The presence of sensors at all the joints enable the robot's control system to be aware of where each link is in the workspace using forward kinematics.

2.2.2 Inverse Kinematics

The converse of forward kinematics, i.e, determining the joint space parameters given the catersian parameters of the end-effector is known as inverse kinematics. This enables us to determine the pose of the manipulator required for the end-effector to reach the required point in its work-space.

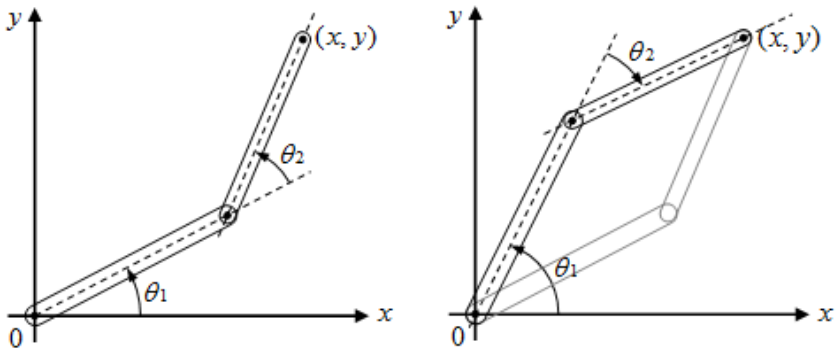


Figure 2.2: Inverse Kinematics Solution for simple 2-DoF manipulator

The inverse kinematic solution gives the possible joint space parameters for a given end-effector coordinate. This means that all the different poses that lead to the end-effector reaching the required point are obtained from the inverse kinematic solution. For the 2-DoF system shown, the two inverse kinematic solutions shown are,

$$\theta_2 = \pm 2 \sqrt{\frac{(L_1^2 + L_2^2)^2 - (x_2^2 + y_2^2)}{(x_2^2 + y_2^2) - (L_1^2 - L_2^2)^2}} \tag{2.1}$$

Extending the same to a multi-DoF system, the general inverse kinematic solution [2] is given by,

$$m = pinv(J) * p \tag{2.2}$$

where, m is the joint space solution, p is the given cartesian coordinate and $pinv(J)$ is the pseudo-inverse of the Jacobian of the transformation matrix T, given by,

$$T = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & x \\ \sin\theta & \cos\theta & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

The pseudo inverse or the Moore-Penrose inverse [3] is given by,

$$pinv(J) = (J^T J)^{-1} J^T \tag{2.4}$$

The pseudo inverse is used here and not the standard matrix inverse operation to make the matrix dimensions comply with the matrix multiplication rule. The transformation matrix in Eq.2.3 is composed of the inner 3×3 rotation matrix and the linear translation vector in the final column.

2.2.3 Trajectory Generation

Once the final pose is calculated from the given end-effector coordinates, a trajectory can be developed based on multiple available modes of traversing available, from the home/resting position of the manipulator. The most commonly used modes of traversing involve generation of a straight line trajectory, a curve-fitted trajectory, or a spline fitted trajectory. The straight line trajectory requires more computation time since making the end-effector move along a straight line is counter intuitive considering most of the primary joints available in the manipulator are revolute joints. The curve-fitted approach involves approximation and there is always an associated error with this approach. Since this is a dynamically changing system and each run is different from the next, a cascading error value may lead to a diverging net error value. This means the manipulator motion will increasingly lose its accuracy at an unpredictable rate and this is certainly not desirable. Spline fitted trajectory traversal is the most time optimized

methods available since it efficiently exploits the presence of revolute joints. This is a counter-intuitive yet efficient method since the splines seems to increase the traversal distance however, a continuous curve means multiple actuators can be active simultaneously, resulting in shorter traversal times. The time optimization is however subject so the spline chosen for the task.

Chosing time 't' as the end-effector path parameter has a second advantage that is, the expressions of all the constraints are simpler and a re-interpolation is not needed after the actuator activation planning. The position vectors of the reference knots P_i are denoted as $\vec{p}_i, i = 0, 1, .., n$ where, P_0 and P_n represent the home and valve positions respectively. The velocity vector at each knot is $\vec{v}_i$ respectively. Let,

$$h_i = t_i - t_{i-1}$$

Then, using Hermite interpolation, the cubic spline equation [6] and thereby the trajectory is given by,

$$\begin{aligned}\vec{C}_i(t) = (\frac{t-t_i}{h_i})^2(2\frac{t-t_{i-1}}{h_i}+1)\vec{p}_{i-1} + (\frac{t-t_i}{h_i})^2(t-t_{i-1})\vec{v}_{i-1} \\ + (\frac{t-t_{i-1}}{h_i})^2(2\frac{t_i-t}{h_i}+1)\vec{p}_{i-1} + (\frac{t-t_{i-1}}{h_i})^2(t-t_i)\vec{v}_i \quad (2.5)\end{aligned}$$

Differentiating the trajectory, $\ddot{C}$, $\dddot{C}$ give the acceleration and jerk vectors respectively. These can be set to a specific threshold value which then helps us determine the boundary equations. Keeping the jerk value to a minimum is better for the longevity of the manipulator joints and links and keeping the acceleration under a particular set limit ensures safety in case a human or animal happens to interfere in the robot's work-space.

2.3 Industrial Manipulator

Industrial Manipulators are mechanisms that involve an assembly of one or more links and joints providing one or higher degrees of freedom for the manipulator to move within its work-space. There is a vast variety of commonly used joints, for e.g., revolute joints, rotational joints, sliding joints e.t.c. Fig.2.3 shows a schematic for a simple 6-link industrial manipulator produced by KUKA, which is a leading robotics company.

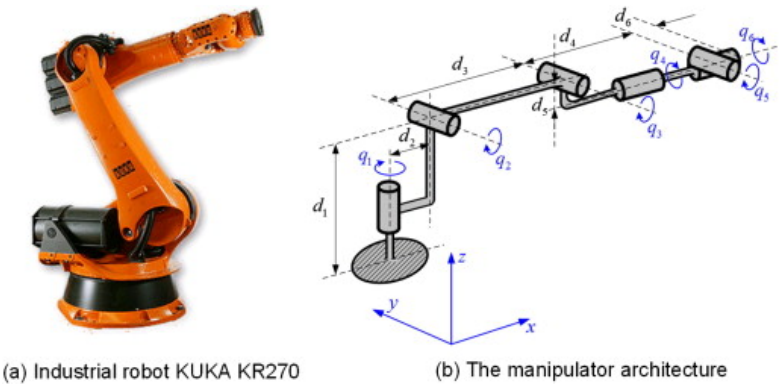


Figure 2.3: Simple 6-link manipulator(left) and its schematic(right)

The work-space is the entire region the manipulator can reach. The available work-space is a subset of the work-space and it is the region the robot is permitted to operate in. Due to safety reasons the available work-space is usually smaller than the net work-space to allow for safety boxes to be built around the manipulator. Fig.2.4 shows the 3D work-space of a standard manipulator. The dexterous work-space is a subset of the work-space in which the manipulator attains no points of singularity, shown in Fig.2.5. The circle labelled '1' is the net work-space and the shaded region represents the dexterous work-space.

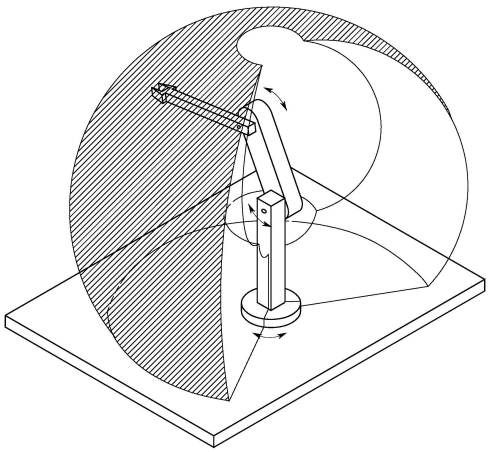


Figure 2.4: 3D work-space of a manipulator

A point of singularity is one at which the manipulator loses one or more degrees of freedom. The most common example of a point of singularity is the toggle position represented by the right most manipulator pose in Fig.2.5.

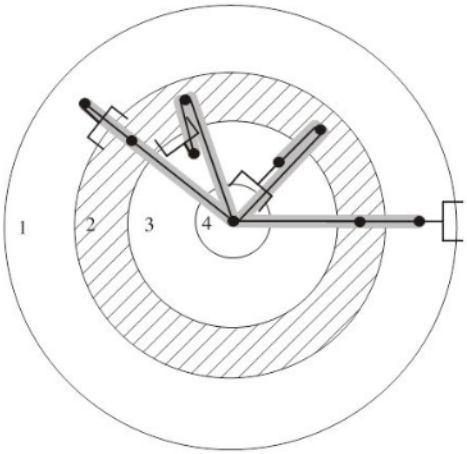


Figure 2.5: Schematic showing the net work-space and the dexterous work-space

The manipulator used for this project is a 6-DoF KUKA KR-16 Robot, shown in Fig.2.6, available at the Siemens Center of Excellence, NIT-Trichy.

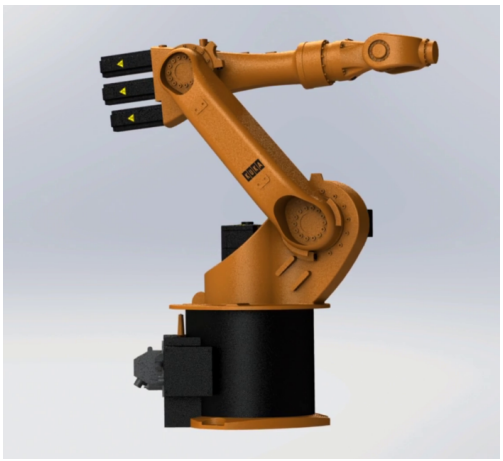


Figure 2.6: KUKA KR-16

Chapter 3

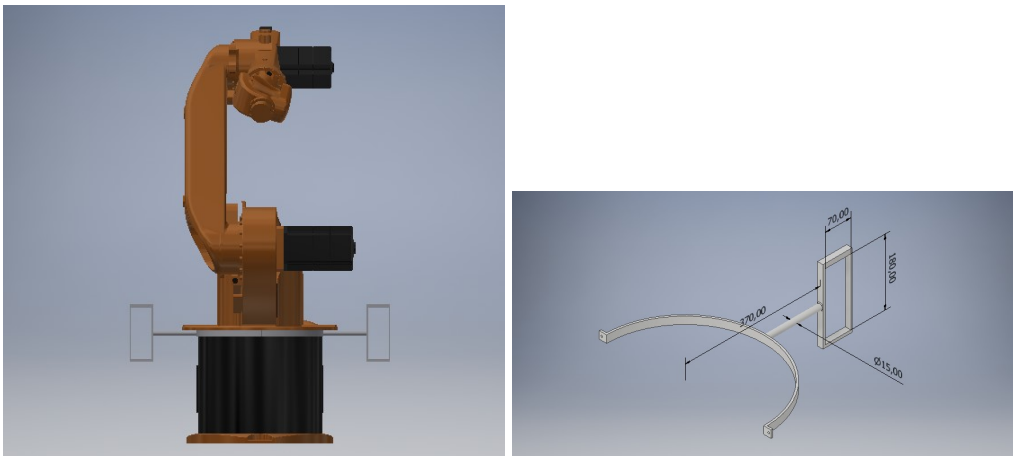
Methodology and Systems Design

3.1 Vision System

The vision system forms one of the most fundamental sub-systems of any autonomous system. A vision system involves the use of one or more sensors to sense the state parameters of the subject and relays it to the control unit which decides whether any changes are required and informs the other sub-systems accordingly. The vision system is the first to detect any change in parameters. The vision system need not always include a camera sensor as the name might suggest. It may consist of temperature sensors, pressure sensors, proximity sensors e.t.c. This project makes use of camera sensors available in most standard low-end phones in the market. The camera sensors used have the following specifications,

1. (16 MP, f/1.7 aperture, 27mm (wide), 1/2.8", 1.12μm, PDAF)$\times$2
2. Standard Low Resolution CCTV Camera

The two camera sensors comprising the stereo camera setup are mounted on the base link of the KR-16 as shown in Fig.3.1. The C-shaped sleeve shown is used to mount the two camera sensors on either side of the base link of the manipulator such that the center of the line joining the two sensors coincides with the origin of the manipulator.



(a) Stereo Camera Clamps (b) C sleeve

Figure 3.1: Stereo Camera Setup

3.1.1 R-CNN

The first objective of the vision system is to detect if a car has been parked in front of the air filling station. To achieve this goal, we have designed a simple R-CNN [4] (Regions for Convolutional Neural Network) that makes use of a standard low resolution CCTV camera to check for cars parked for tire inflation. Since each driver might park the car in a slightly different position and orientation, the Neural Network must be robust enough to detect whether the car is stationary for tire inflation, or if a car is just passing by that area. For this, the Network should first be capable of distinguishing a car from other objects such as people, plants, animals/birds, motorbikes e.t.c. The chosen Neural Network is an R-CNN to tackle this problem since it is efficient in detecting objects in an image and classifying them in a time-optimized manner into the classes it is trained with. To facilitate this, the region where the car is expected to stop is painted with yellow floor paint indicating the driver to park in this region as shown in Fig.3.2. The system is however designed to work even if the vehicle is partially or fully outside the painted region. The first manipulator is designated to be positioned to the left of the air pump and the second manipulator is positioned opposite the first.

Figure 3.2: Parking region for filling air at a petrol bunk

Architecture

The R-CNN is built of an input layer fed through an convolutional hidden layer passing through a BBox (Bounding Box) layer and a SVM (Support Vector Machine) layer which finally classifies the image features into one of the output classes namely, car, human, animal, bird, carR, aeroplane, monitor screen. Here, the 'carR' class refers to the rear end of a car. The architecture is shown in Fig.3.3.

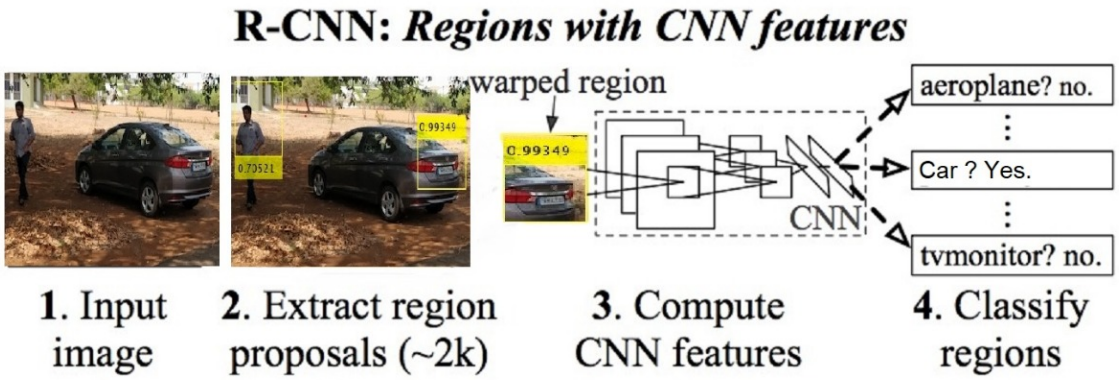


Figure 3.3: R-CNN Architecture

Training and Detection

The R-CNN is trained with an online database of 296 test images and biases are generated corresponding to the 'carR' class. The BBox is the layer responsible for classifying sections of the image with features of varying strength that have a probability of being classified as one of the output classes. This layer is biased to classify cars and the rear end of cars into two separate bounding boxes. The BBox is also responsible for resizing the box in order to assume minimum area while including the strongest features as in-liers. The SVM is responsible for categorizing the images based on its bias value. It does so by maximizing the marginal difference value between two different output classes, thereby giving classification results with high accuracy. The hidden layer architecture is shown in Fig.3.4.

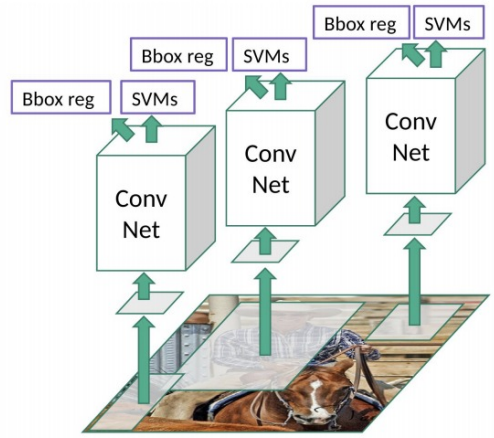
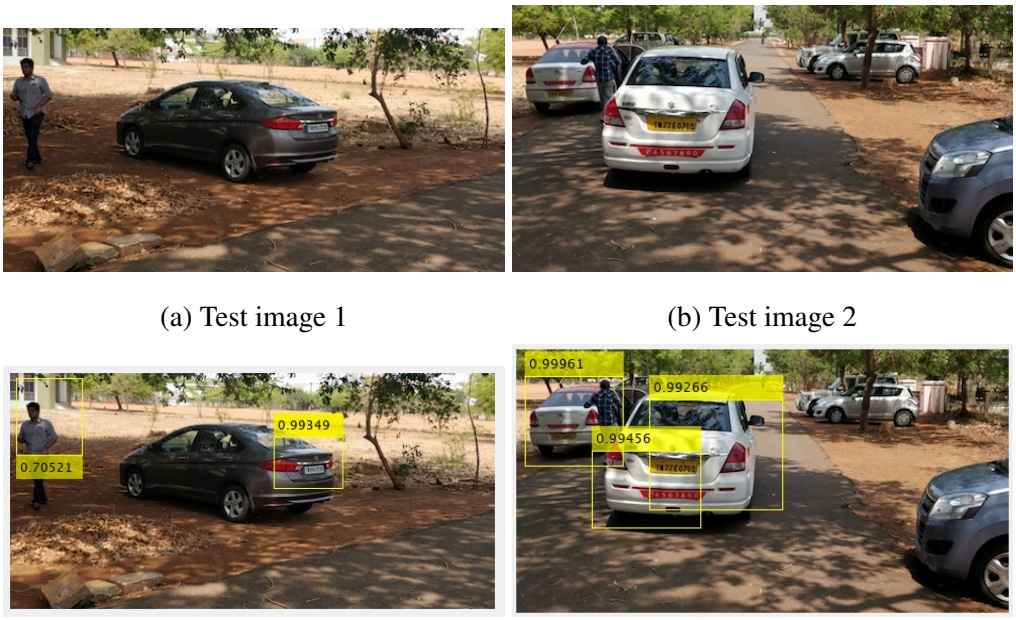


Figure 3.4: Hidden Layer Architecture

R-CNN Detection Results

The R-CNN code (Appendix.A.1) was tested on the above images take on Institute Road at NIT, Trichy and the respective results are shown in Fig.3.5.

The value associated with each detection box represents the probability for that box being classified as a car rear and the box with highest probability is an accurate detection of a car rear in all the 35 test cases.



(a) Test image 1

(b) Test image 2

(c) Detected box image 1

(d) Detected box image 2

Figure 3.5: R-CNN Test Results

3.1.2 Valve Detection

Once the R-CNN code detects a car rear parked inside or partially inside the marked region, it triggers the system to execute the valve detection code. The valve detection has two major parts,

1. Calculating distance of the tire from robot center
2. Determining valve position and orientation

Tire Detection

The first step to be performed once the valve detection code is triggered is to detect the car tire. Detection of the tire involves,

1. Detecting the tire outer and rim boundaries
2. Obtaining inner tire diameter
3. Detecting tire center

The test image used for this section is shown in Fig.3.6

Figure 3.6: Test Image

To process the image, its pixel matrix values are first normalized with a suitable threshold, resulting in a pixel matrix with matrix entry values ranging from $0 - 1$. This is essentially a black and white image of the original image captured by the stereo camera system shown in Fig.3.1.

Next a Dilated Gradient Mask is applied which serves as a Gaussian smoothening filter in order to facilitate the edge detection algorithm (Appendix.A.2). Fig.3.7 shows the smoothened image with the distinctly determined tire edges.

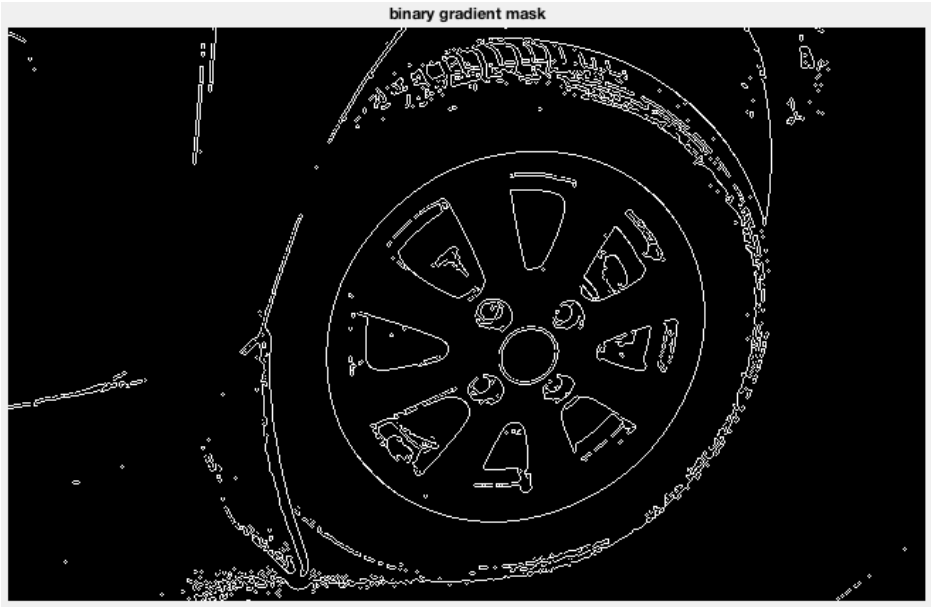
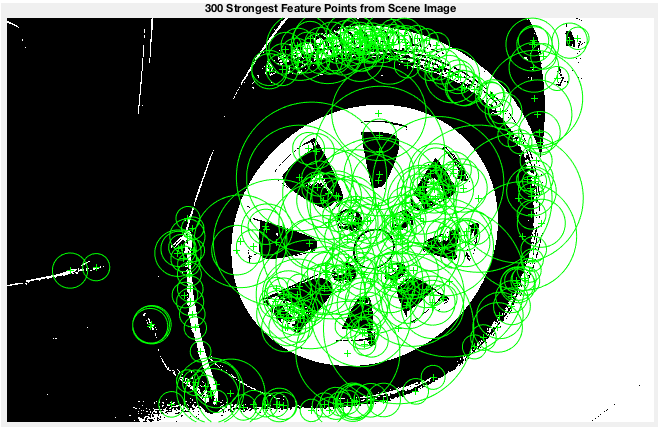


Figure 3.7: Dilated Gradient Mask with detected tire edges

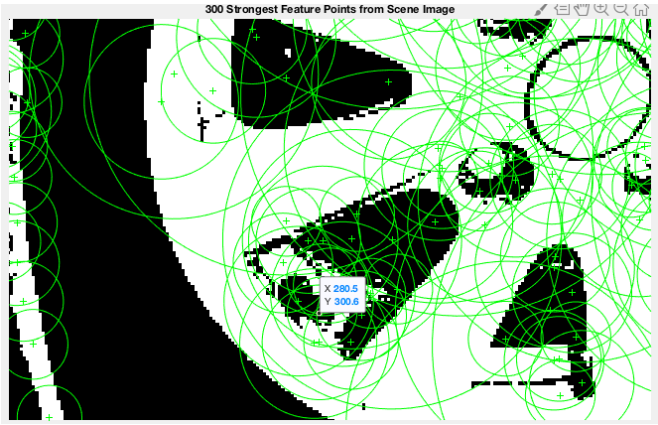
Next, The holes inside the detected tire edges are filled to enable calculation of the inner diameter as shown in Fig.3.8

Figure 3.8: Filled holes for diameter calculation

Valve Orientation



(a) 300 Strongest Surf Features



(b) Close up of the tire valve feature

Figure 3.9: SURF Features

To detect the valve, we need to detect the strongest features in the image and compare it with a standard image to match similar features. The matching parameters are biased to converge on detecting the valve in minimum possible iterations. In order to extract features, we make use of SURF [5] (Speeded Up Robust Features) extraction technique rather than SIFT (Scale Invariant Feature Transform) since SURF is time-optimized and our scene image does not have too many scale variations. The SURF features of the test image are shown in Fig.3.9. The center of each of the green circles represents an extracted feature and the diameter of the circle represents the strength of that feature. The coordinates of the valve can now be manually obtained from Fig.3.9(b). However, the system needs to function autonomously which implies the valve detection is not complete. For the system to detect the valve, we have to compare our scene image with a standard tire image. On comparing their SURF features and matching corresponding features, we should be able to detect the valve with an associated error value. The matches formed are putative matches since they have an associated probabilistic error.

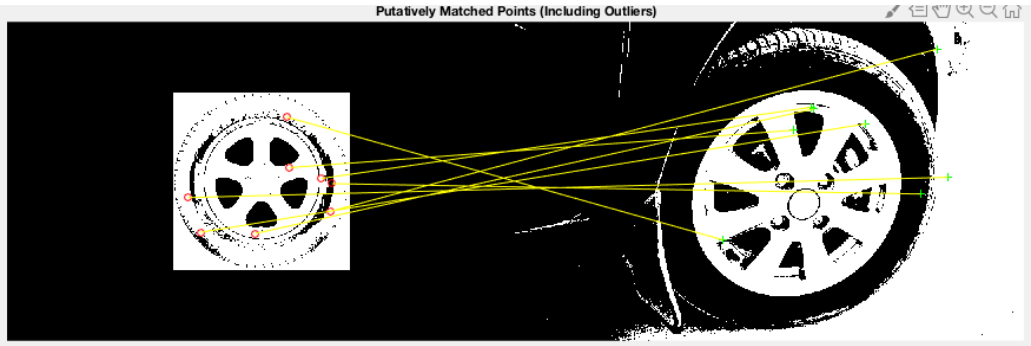


Figure 3.10: Putatively matched points

The putative matches formed on the outliers of the scene image, are superimposed on the putative matches on the inliers to give putative matches (Fig.3.10) of highest probability. The region among these with highest probability is labelled the Detected Box (Fig.3.11). On repeating the iterative process with the Detected Box of one iteration as the scene image for the next, we converge at the valve with an associated error value of $(\pm 5 - 35mm)$.

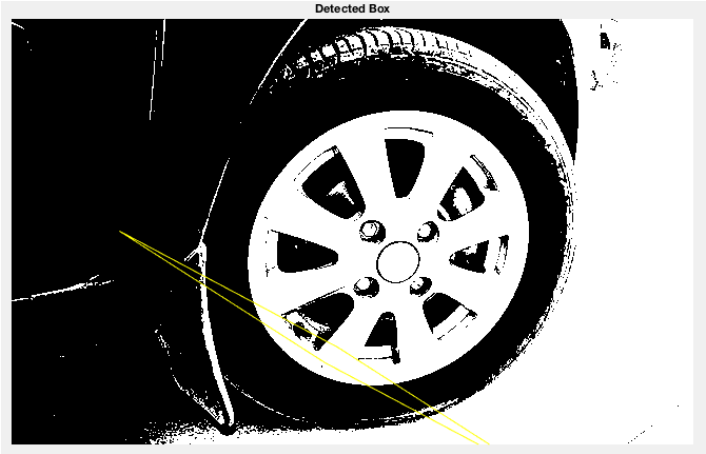


Figure 3.11: Detected Box, first iteration

Extracting Tire Center Coordinates

Having detected the valve, we know the position and orientation of the valve w.r.t the center of the tire. Calculating the coordinates of the tire center from the origin of the manipulator will give us all the required coordinates to feed to the motion algorithm. To calculate the distance of the center from the camera setup, we use the triangulation method. As explained in Fig.3.1 the center of the camera sensor line coincides with the origin of the robot. Thus, finding the coordinates of the center of the tire using the center of the camera line as the origin will suffice. To do this, we use the triangulation formula for the two images of the same scene captured by each sensor of the stereo camera setup. The tire center feature extracted in each image is matched and this forms the reference line for triangulation as shown in Fig.3.12.

The 'x, y' coordinates are simply the pixel value of the tire center. The distance of the required point from the focal plane is the 'z' coordinate and is given by Eq.3.1.

$$z = b\frac{f}{x_l - x_r} \tag{3.1}$$

Here, 'b' is the distance between the two cameras, 'f' is the distance pf the cameras from the focal plane that is determined using the reference line, 'x_l, x_r' are the pixel coordinates of each of the left and right images respectively. Thus, the final valve co-ordinates with respect to the origin of the manipulator are calculated and passed to the motion algorithm explained in Sec.3.2

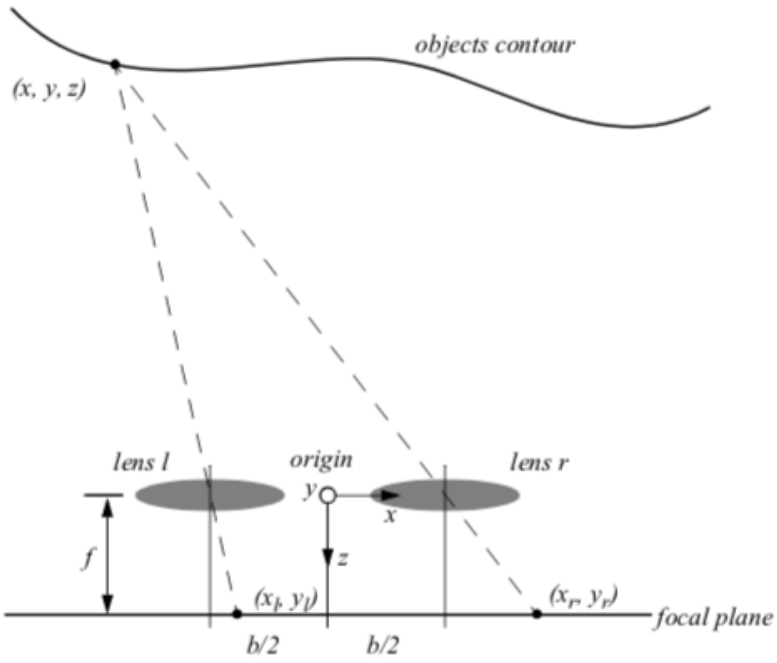


Figure 3.12: Triangulation Method for Depth Calculation

3.2 Motion Algorithm

After obtaining the final valve coordinates w.r.t the origin of the manipulator, the final pose of the manipulator is calculated using Inverse Kinematics as explained in Sec.2.2.2, from Eq.2.2. This gives us the joint parameters to be achieved for the end-effector to reach the valve coordinates. The spline-fitted trajectory to reach this point, from the manipulator's home position, is generated as explained in Sec.2.2.3 using Eq.2.5. The standard P2P mode (Point to Point) can be used to achieve this, but this project explores a new methodology in order to achieve added time-optimization. For this application, we make the assumption that the car is parked in or near the designated region shown in Fig.3.2. This assumption enables us to restrict the manipulator's work-space to an area with an $85°$ FoV (Field of View) on either side of the Home Position of the manipulator. This restricted work-space permits the robot to reach any point near or in the designated zone without leaving its dexterous work-space (explained in Sec.2.3). This means, to calculate the final pose, our algorithm need not determine solutions with points of singularity and avoid those solutions since any solution within the dexterous space is void of singularities. The custom function *'ikine6s()'* (Appendix.B.1) performs this task of restricting the available work-space to the dexterous work-space, thereby increasing the compactness of the code and marginally decreasing computing time. Restriction of the

work-space also results in added human-robot safety parameters which is alwayas a considerable advantage for fully autonomous systems. A simulation is first run on the computer to check for correctness, after which the codes are run on the manipulator.

3.3 Local Network Establishment

The vision and motion algorithms are coded on MATLAB to increase the ease of access to the code and facilitate code updates. To make the MATLAB codes compatible with the KR-16 manipulator we need to set up a local network and connect both the computer and the manipulator onto a common host as shown in Fig.3.13. This enables us to by-pass the KSS (KUKA System Software) that runs its own programming language, KRL (KUKA Robot Language). This allows us to make modifications to motion codes and enables us to perform the work-space restriction in order to move away from the in built P2P codes that are not time-optimized. To do this, we use the $X66$ (Fig.3.14) ethernet port available on the KRC (KUKA Robot Controller). Having established a host network and connecting both the computer and the KRC to it, we directly make use of the KCT (KUKA Control Toolbox) that enables the MATLAB code to by-pass the KRL compiler and run directly on the KR-16. The experimental setup of the Network connection is shown in Fig.3.15

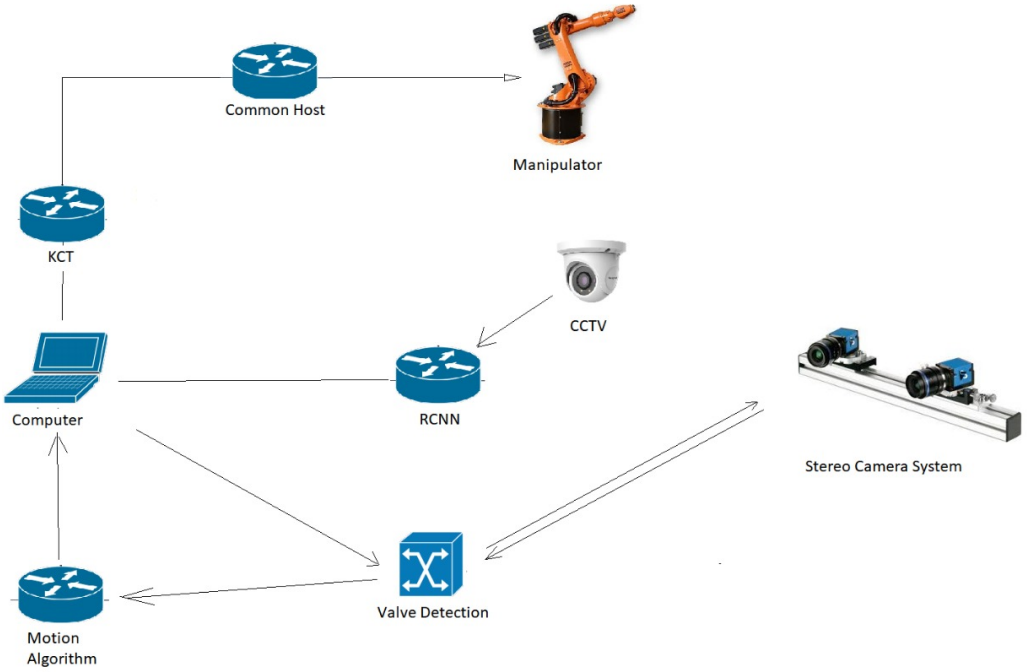


Figure 3.13: Network Schematic

Figure 3.14: X66 Port on the KRC

Figure 3.15: Experimental Network Setup

Chapter 4

Experimentation and Results

4.1 Experimental Setup

The experimental setup (Fig.4.1) consists of the KR-16 enclosed in a safety box, and a vehicle tire placed inside the work-space manually at multiple random positions and orientations. Since the C-sleeve could not be fit onto the manipulator owing to the manipulator being used in another experiment simultaneously, the stereo camera setup was moved along the line passing through the origin of the manipulator, to a distance of $5m$ such that the camera system was outside the safety box as shown in Fig.4.2. The three random positions shown were tested for and validated through a continuous video, and the manipulator reached the valve position each time with the prescribed $\pm 5 - 35mm$ accuracy measured by the clamp laser sensor mounted on the end-effector.

Figure 4.1: Experimental Setup

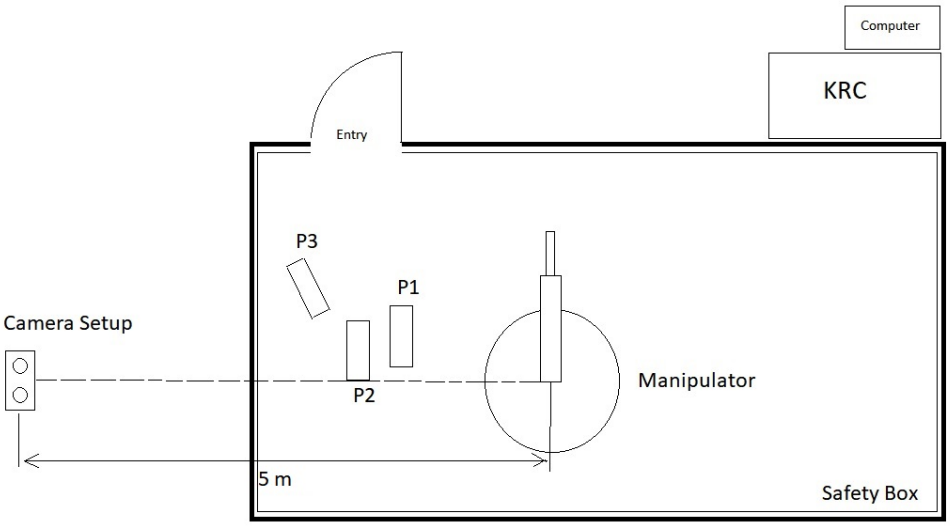


Figure 4.2: Schematic of Experimental Setup

4.2 Vision System Results

The tire position in the first run was as shown in Fig.4.3

Figure 4.3: Tire position on first run

The processed images are as shown in Figs.4.4, 4.5, 4.6

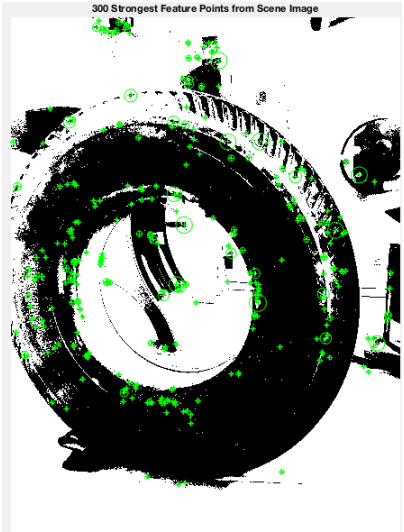
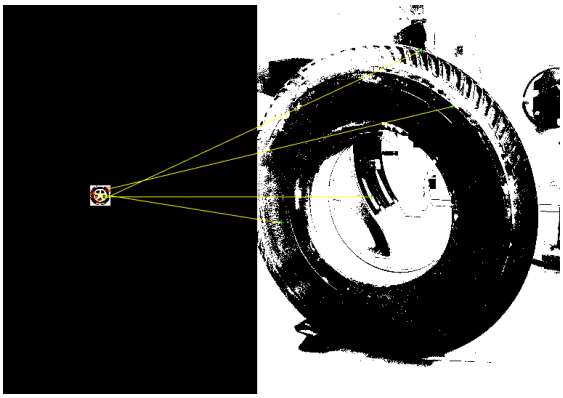


Figure 4.4: SURF features for first run

(a) Outlier putative matches for first run



(b) Inlier putative matches for first run

Figure 4.5: Experimental putative matches for first run

4.3 Motion Algorithm Results

The results are shown in Figs. 4.7, 4.8, 4.9,

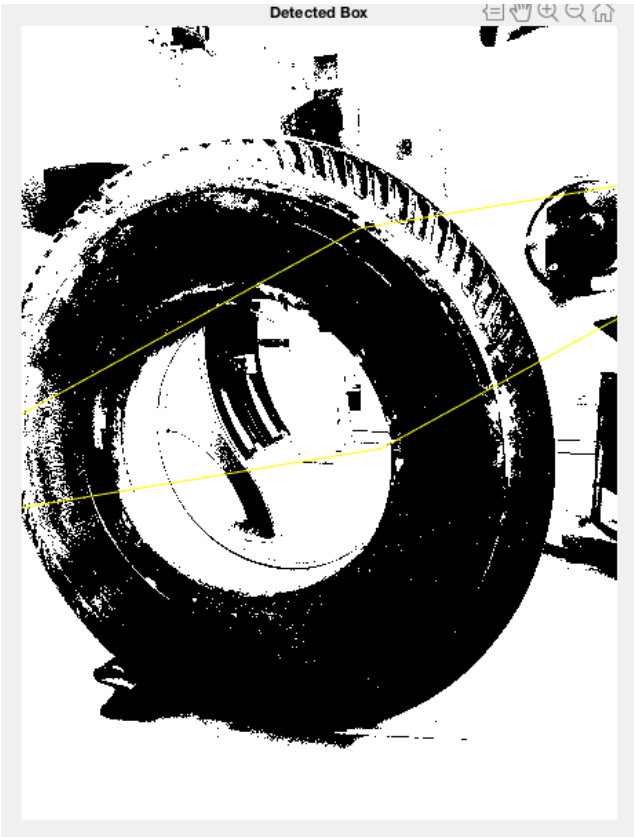
The positions P_1, P_2, P_3 shown in Fig.4.2 were placed at random and were tested for using the setup shown, in one continuous attempt. The parameters generated by the system are listed in Tab.4.1. Here, θ is the angle made by the projection of the horizontal diameter of the tire on the $X-Y$ plane w.r.t the manipulator's $X-Axis$ which is the horizontal axis in the schematic shown in Fig.4.2 and the vertical axis is the manipulator $Y-Axis$. ϕ is the angular orientation of the valve w.r.t the tire's vertical axis. The error associated with position parameters is $\pm 5-35mm$

Run number	Position (x, y, z) in 'm'	Orientation ($\theta \pm 2.5°, \phi \pm 3°$)
1	(-2.358,1.734,0.746)	(87.6°,14°)
2	(-2.738,1.266,0.468)	(88.2°,106°)
3	(-3.421,2.381,0.391)	(121.7°,253°)

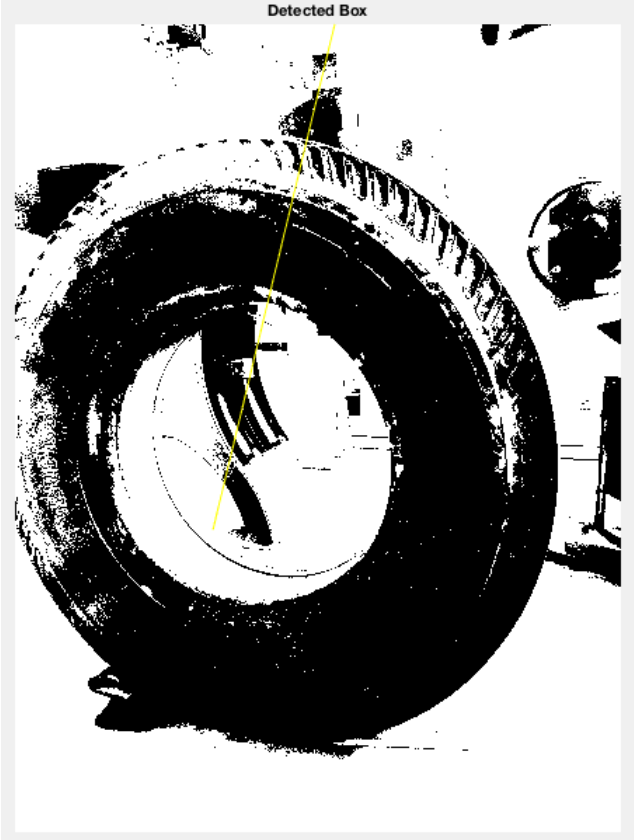
Table 4.1: Experimental Results

Once the gripper attached to the end-effector reaches the valve position with a maximum error of $35mm$, it closes its prongs around the valve and the tactile centers close the prongs until both prongs are in contact with the subject. This ensures that despite a comparatively large initial error, the final gripper position is exactly at the valve.

The non-collision and the human-robot safety algorithm layered onto the robot are in-built functions that cannot be by-passed owing to safety reasons and need not be since they do not interfere with the motion algorithms.



(a) Detected Box in first iteration



(b) Detected Box in second iteration

Figure 4.6: Detected Box for first run

Figure 4.7: First run result

Figure 4.8: Second run result

Figure 4.9: Third run result

Chapter 5

Summary and Conclusions

5.1 Summary

The task of autonomous tire inflation may seem a straight forward application of existing manipulator technology. This project proves that to work in a dynamically changing environment, a non-autonomous system has to be augmented with subsidiary systems that are capable of sensing state changes, formulating a new plan of action and implementing it. The vision system is capable of identifying proper or improper vehicle positioning, and detecting the valve coordinates with work-able accuracy in a dynamic environment. The motion algorithms serve the purpose of being a time-efficient method of moving the end-effector to the desired position while adding extra safety features, since the chances for human-robot interaction are extremely high in a fully autonomous system setup.

5.2 Conclusions

The versatility of the industrial manipulator was proved to be applicable to dynamic, non-controlled environments as well apart from the controlled factory environment. However, each application requires a separate set of systems and algorithms tailor-made for that application as opposed to a general set of in-built functions that can be used for a wide rage of applications in a controlled factory environment. Such is the case however, with any fully autonomous system that needs to be trained efficiently enough to cope with unpredictable changes. With the future of technology moving towards auton-

omy in systems, the fact that a system built to work as a non-autonomous component can be tweaked to perform almost as efficiently in an autonomous environment as well is highly advantageous. The vision system and manipulator motion algorithms form a cost effective system that is a solution to the given problem statement. The study on economic feasibility in the Indian market is however, out of the scope of this project.

5.3 Scope for Future Work

This project deals with tire valve detection and generating a time optimized method to move an industrial manipulator to the desired location, in order to begin the inflation. The process of inflation that involves the following can be pursued in future projects.

1. Designing a gripper to unscrew the valve cap and attach the pipe from the pump to the valve
2. Method to measure the current tire pressure and increase or decrease it to the optimum levels based on the type of vehicle detected

5.4 Other Areas of Application

The underlying aim of this project is to augment an open-loop device such as an industrial manipulator with subsidiary systems to make it a fully autonomous system capable of working in a dynamic environment. Taking this principal further, a few of the many other possible areas of application include,

1. Autonomous vehicle repair systems
2. Autonomous surgery/ other Bio-medical applications
3. Autonomous cleaning systems for oil spills/ oceanic, land debris
4. Autonomous maintenance systems in inaccessible locations such as nuclear reactors, particle accelerators e.t.c

References

[1] Introduction to robotics: mechanics and control, 3/E, Craig, John J. Pearson Education India, 2009.

[2] Introduction to autonomous mobile robots, Siegwart, Roland and Nourbakhsh, Illah Reza and Scaramuzza, Davide. MIT Press, 2011.

[3] Robot dynamics and control, Spong, Mark W and Vidyasagar, Mathukumalli. John Wiley & Sons, 2008

[4] Fast r-cnn, Girshick, Ross. Proceedings of the IEEE international conference on computer vision, p1440–1448, 2015

[5] Surf: Speeded up robust features, Bay, Herbert and Tuytelaars, Tinne and Van Gool, Luc. European conference on computer vision, p404–417, 2006

[6] Cubic spline trajectory generation with axis jerk and tracking error constraints, Zhang, Ke and Guo, Jian-Xin and Gao, Xiao-Shan. International Journal of Precision Engineering and Manufacturing, 14/7-p1141–1146, Springer Publications, 2013

Appendices

Appendix A

MATLAB CODE- Vision System

A.1 R-CNN

```
  % Load vehicle data set
data = load('fasterRCNNVehicleTrainingData.mat');
vehicleDataset = data.vehicleTrainingData;

% Display first few rows of the data set.
vehicleDataset(1:4,:)

% Add fullpath to the local vehicle data folder.
dataDir = fullfile(toolboxdir('vision'),'visiondata');
vehicleDataset.imageFilename = fullfile(dataDir, ...
    vehicleDataset.imageFilename);

% Read one of the images.
I = imread(vehicleDataset.imageFilename{10});

% Insert the ROI labels.
I = insertShape(I, 'Rectangle', vehicleDataset.vehicle{10});

% Resize and display image.
I = imresize(I,3);
figure
imshow(I)
```

```
22
23 % Split data into a training and test set.
24 idx = floor(0.6 * height(vehicleDataset));
25 trainingData = vehicleDataset(1:idx,:);
26 testData = vehicleDataset(idx:end,:);
27
28 % Create image input layer.
29 inputLayer = imageInputLayer([32 32 3]);
30
31 % Define the convolutional layer parameters.
32 filterSize = [3 3];
33 numFilters = 32;
34
35 % Create the middle layers.
36 middleLayers = [
37
38     convolution2dLayer(filterSize, numFilters, 'Padding', 1)
39     reluLayer()
40     convolution2dLayer(filterSize, numFilters, 'Padding', 1)
41     reluLayer()
42     maxPooling2dLayer(3, 'Stride',2)
43
44     ];
45
46 finalLayers = [
47
48     % Add a fully connected layer with 64 output neurons. The ...
          output size
49     % of this layer will be an array with a length of 64.
50     fullyConnectedLayer(64)
51
52     % Add a ReLU non-linearity.
53     reluLayer()
54
55     % Add the last fully connected layer. At this point, the ...
          network must
56     % produce outputs that can be used to measure whether the ...
          input image
```

```
    % belongs to one of the object classes or background. This ...
        measurement
    % is made using the subsequent loss layers.
    fullyConnectedLayer(width(vehicleDataset))

    % Add the softmax loss layer and classification layer.
    softmaxLayer()
    classificationLayer()
];

layers = [
    inputLayer
    middleLayers
    finalLayers
    ]

% Options for step 1.
optionsStage1 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 1, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

% Options for step 2.
optionsStage2 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 1, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

% Options for step 3.
optionsStage3 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 1, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

% Options for step 4.
optionsStage4 = trainingOptions('sgdm', ...
```

```
95      'MaxEpochs', 10, ...
96      'MiniBatchSize', 1, ...
97      'InitialLearnRate', 1e-3, ...
98      'CheckpointPath', tempdir);
99
100 options = [
101     optionsStage1
102     optionsStage2
103     optionsStage3
104     optionsStage4
105     ];
106
107 % A trained network is loaded from disk to save time when ...
        running the
108 % example. Set this flag to true to train the network.
109 doTrainingAndEval = false;
110
111 if doTrainingAndEval
112     % Set random seed to ensure example training reproducibility.
113     rng(0);
114
115     % Train Faster R-CNN detector. Select a BoxPyramidScale of ...
            1.2 to allow
116     % for finer resolution for multiscale object detection.
117     detector = trainFasterRCNNObjectDetector(trainingData, ...
            layers, options, ...
118         'NegativeOverlapRange', [0 0.3], ...
119         'PositiveOverlapRange', [0.6 1], ...
120         'NumRegionsToSample', [256 128 256 128], ...
121         'BoxPyramidScale', 1.2);
122 else
123     % Load pretrained detector for the example.
124     detector = data.detector;
125 end
126
127 % Read a test image.
128 I = imread('carp.png');
129
130 % Run the detector.
```

```
[bboxes,scores] = detect(detector,I);

% Annotate detections in the image.
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure
imshow(I)
```

A.2 Valve Detection

```
  I = imread('tyreiter.jpg');

BW1 = im2bw(I,0.355);

Iseg = cellDetectionMATLAB(BW1);

% Display the original image and the segmented image side-by-side.
% imshowpair(BW1,Iseg,'montage')

function BWfinal = cellDetectionMATLAB(I)
%cellDetectionMATLAB - detect cells using image segmentation. ...
    We require
%Images 1, 3, 4/5.

[¬, threshold] = edge(I, 'sobel');
fudgeFactor = .5;
BWs = edge(I,'sobel', threshold * fudgeFactor);
figure
imshow(BWs)
title('binary gradient mask');

se90 = strel('line', 3, 90);
se0 = strel('line', 3, 0);

BWsdil = imdilate(BWs, [se90 se0]);
figure
imshow(BWsdil)
```

```
title('dilated gradient mask');

BWdfill = imfill(BWsdil, 'holes');
figure
imshow(BWdfill);
title('binary image with filled holes');

BWnobord = imclearborder(BWdfill, 4);
figure
imshow(BWnobord)
title('cleared border image');

seD = strel('diamond',1);
BWfinal = imerode(BWnobord,seD);
BWfinal = imerode(BWfinal,seD);
figure
imshow(BWfinal)
title('segmented image');

bimg = imread('testtyre.jpeg');
boxImage = im2bw(bimg,0.3);

boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(I);

figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));

figure;
imshow(I);
title('300 Strongest Feature Points from Scene Image');
hold on;
plot(selectStrongest(scenePoints, 300));

[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(I, scenePoints);
```

```
66
67 boxPairs = matchFeatures(boxFeatures, sceneFeatures);
68
69 matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
70 matchedScenePoints = scenePoints(boxPairs(:, 2), :);
71 figure;
72 showMatchedFeatures(boxImage, I, matchedBoxPoints, ...
73     matchedScenePoints, 'montage');
74 title('Putatively Matched Points (Including Outliers)');
75
76 [tform, inlierBoxPoints, inlierScenePoints] = ...
77     estimateGeometricTransform(matchedBoxPoints, ...
         matchedScenePoints, 'affine');
78
79 figure;
80 showMatchedFeatures(boxImage, I, inlierBoxPoints, ...
81     inlierScenePoints, 'montage');
82 title('Matched Points (Inliers Only)');
83
84 boxPolygon = [1, 1;...                           % top-left
85         size(boxImage, 2), 1;...                 % top-right
86         size(boxImage, 2), size(boxImage, 1);... % bottom-right
87         1, size(boxImage, 1);...                 % bottom-left
88         1, 1];                   % top-left again
89
90 newBoxPolygon = transformPointsForward(tform, boxPolygon);
91
92 figure;
93 imshow(I);
94 hold on;
95 line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
96 title('Detected Box');
```

Appendix B

MATLAB Code- Motion Control

B.1 Motion Control

```
1
2  mdl_puma560
3
4  q = [0 -pi/4 -pi/4 0 pi/8 0]
5
6  T = p560.fkine(q)
7
8  qi = p560.ikine(T, 'pinv');
9
10 qi = p560.ikine6s(T)
11
12 p560.fkine(qi)
13
14 p560.ikine6s(T, 'rdf')
15
16 T1 = transl(x1, y1, z1) % end point
17
18 T2 = transl(x2, y2, z2) % start point
19
20 T = ctraj(T2, T1, 15);  % compute cubic spline trajectory
21
22 q = p560.ikine6s(T);
```

```
23
24 subplot(3,1,1); plot(q(:,1)); xlabel('Time (s)'); ...
       ylabel('Joint 1 (rad)');
25
26 subplot(3,1,2); plot(q(:,2)); xlabel('Time (s)'); ...
       ylabel('Joint 2 (rad)');
27
28 subplot(3,1,3); plot(q(:,3)); xlabel('Time (s)'); ...
       ylabel('Joint 3 (rad)');
29
30 clf
31
32 plot(T1)
33 p560.plot(q)
```