Vision-Based Control of Industrial Robot Arm

31/10/2019

Muhammad Hasham Khan[1]

[1] *Student of Mechatronics Engineering,*
The University of Newcastle, Callaghan, NSW 2308, AUSTRALIA
Student Number: 3256011
E-mail: c3256011@uon.edu.au

Executive summary

This project aims to demonstrate a visual servoing system for an industrial robotic manipulator and its practical use. Vision based control is a growing field in the robotic world being used to guide autonomous vehicles and control industrial manipulators. This project explores the vision based control approach to develop a control scheme that uses a camera to guide the end effector of the ABB robot arm and an algorithm to classify objects.

This task involves developing a control scheme for visual servoing, training an object classifier, designing a minimal path planner, demonstrating the manipulator in simulation and operating the robot to reach a desired position for the end effector. The simulation and hardware demonstration task will be split between part A and part B.

Future work for this project involves performing a hardware demonstration using simulation environments and experiments with the ABB robot.

Acknowledgements

You may like to say thank you to someone that helped you with your project.

Contents

1 Introduction

1.1 Background

Robotic manipulators are very useful since they can perform precise, repetitive and dangerous tasks that humans cannot because of their speed, accuracy and reliability. Due to their benefits, they are used to automate tasks such as welding [CITE ME], painting vehicles [CITE ME] and even harvesting crops CITE ME[]

Automation of robotic manipulatos has been done in manufacturing, painting cars, gimbals etc using different kinds of sensors such as IMU's and laser range finders [cite me]

A very useful technique is using cameras to performs this control. It uses less sensors, can be more economical, can use sensor fusion to combine multiple cameras. This is especially useful when the robotic manipulator is not in a predefined environment. An example is the robot arm on the mars curiosity rover [cite me]

This technique is known as visual servoing, and it has two commonly used techniques.
One is to use a fixed point camera that views the object in space while another is a camera attached to the end effector.
here are the benefits of each:
- eye in hand results in closed loop control while fixed camera is an open loop system

1.2 Problem statement

The goal of this project is to use an end effector camera to perform a common automation task used in industry; to sort and pick and place predifined objects using visual control using a industrial robotic manipulator.

exampes of such a task are harvesting robots, car painting robots in factories and the mars rover arm

ASSUMPTIONS
This task involves sorting predefined objects on a known workspace into known locations. The predefined objects need to be of uniform shape, be visually differentiable and be easily movable by the arm
The location bins may be a marked location on the workspace
The workspace itself will be a flat table surface free of obstructions and with suitable lighting

- To perform this task, we need to:
1 Classify objects to be sorted
2 Locate objects to be sorted using a camera
3 Generate a trajectory to sort items
4 Control a robotic manipulator to reach desired end effector pose's // follow a path
5 Design a visual servoing system to control the robot

1.3 Literature Review

- 1: classifying objects in an image has been done using ”examples”.
can classify an image or you can classify the location the object is in the image.
there have been many recent advances in the field, e.g. alexnet popularising the cnn architecture and the subsequent boost in the use of GPU’s.
Talk about RCNN, fast RCNN and faster RCNN and then that vs SSD vs YOLO vs RESNET etc.
The decision to use tensorflows open source libraries vs matlab.
Ultimately, Faster RCNN was chosen as it provides the greatest accuracy at the cost of greater computing time. This was not an issue for this project as speeds will be limited

- 2: Locating objects is done using line detection and feature extraction. These features can then be manipulated to find the location of the object relative to the end effector position or used directly to control the movement of the arm.

- 3: Items can be arbitrarily placed on a work surface and the minimal path to each points needs to be generated. This can be seen as a subset of the travelling salesman problem, whereby each point can be seen as node that can only be visited once.
The soluton can then be used to generate a path as a series of discrete poses in the operation space for the end effector to follow.

- 4: This component requires a simulation model of the robotic manipulator and forward and inverse kinematics to follow a desired trajectory.
- 5: This component requires interfacing with the manipulator through a controller, interfacing with the camera and getting images and generating a control action. PBVS vs LBVS

1.4 Motivation

the use of machine vision systems for control automisation is part of modern control structures.
this project aims to demonstrate and implement these techniques on a real platform. wants to integrate vision control with industrial robot. to select classify and manipulate objects.
This is a benchmark but these techniques can be implemented/adapted to other tasks more relevant to industry.
integration of vision within robotics was not part of my degree and was useful to learn

2 Robotic Manipulator Simulation

To perform a pick and place task, a suitable robotic manipulator was required. The ABB IRB 120 robotic manipulator was provided by the University of Newcastle to be used:

2.1 ABB IRB 120

- introduce the robot and its specifications
The IRB 120 is a 6DOF robotic manipulator commonly used in industry due to it's small size and high manuevarability. It can carry a payload of 3Kg at it's end effector, can reach 580mm horizontally from it's base and an end effector accuracy of 0.5mm. This makes it ideal for the desired pick and place task
- The anthropomorphic structure of the ABB robot arm mimics the functionality of a real human arm, albeit with one less degree of freedom.
- the desired end effector is not available and will be chosen or designed in part B. Such a system's design criterion and subsequent impacts with be further discussed in section XX
- To use the robotic arm, a simulation model was required

2.2 Modelling

Modelling of the robot can be done as an open serial kinematic chain. This involves defining a co-ordinate system for each joint of the
The robot consists of 6 joints that can move freely about an axis, as shown in Figure 1

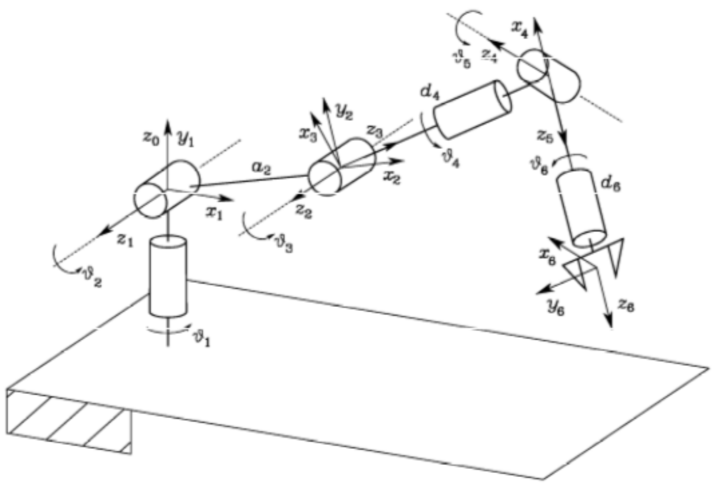

Figure 1: an Anthropomorphic arm, from Siciliano

One can then perform co-ordinate transformations to define a subsequent joint in terms of a rotation about the prior joint.
The Denavit Hartenberg convention was used to simplify many of these transformations
It works by standardising revolutions and prismatic extensions to only occur about the z axis.

The Denavit hartenberg paramaters were used and found at https://robodk.com/Python-API-examples

Table 1: Denavit Hartenberg parameters for ABB IRB 120

	q	**d**	**a**	**alpha**
Joint 1	0	0	290	-pi/2
Joint 2	-pi/2	270	0	0
Joint 3	0	70	0	-pi/2
Joint 4	0	0	302	pi/2
Joint 5	pi/2	0	0	-pi/2
Joint 6	pi	0	72	0

The transformation matrix is composed using

$$[h]\,{}^{N-1}T_N = Rot(q)Trans(d)Rot(a)Trans(\alpha) \tag{2.1}$$

Which results in the following co-ordinate transformation matrix describing the current joint with respect to the previous

$$^{n-1}\mathbf{T_n} = \begin{bmatrix} cos\theta_n & -sin\theta_n cos\alpha_n & sin\theta_n cos\alpha_n & r_n cos\theta_n \\ sin\theta_n & cos\theta_n cos\alpha_n & -cos\theta_n sin\alpha_n & r_n sin\theta_n \\ 0 & sin\alpha_n & cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2.3 Forward Kinematics

Using the parameters from Table 1, the following Transformation matrices were constructed:

$$^{0}\mathbf{A_1} = \begin{bmatrix} cosq_1 & -sinq_1 & 0 & 0 \\ sinq_1 & cosq_1 & 0 & 0 \\ 0 & 0 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$^{1}\mathbf{A_2} = \begin{bmatrix} cosq_2 & 0 & -sinq_2 & 0 \\ sinq_2 & 0 & cosq_2 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$^{2}\mathbf{A_3} = \begin{bmatrix} cosq_3 & -sinq_3 & 0 & a_3 cosq_3 \\ sinq_3 & cosq_3 & 0 & a_3 sinq_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$^{3}\mathbf{A_4} = \begin{bmatrix} cosq_4 & 0 & -sinq_4 & a_4 cosq_4 \\ sinq_4 & 0 & cosq_4 & a_4 sinq_4 \\ 0 & -1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$^{4}\mathbf{A_5} = \begin{bmatrix} cosq_5 & 0 & -sinq_5 & 0 \\ sinq_5 & 0 & -cosq_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$^{5}\mathbf{A_6} = \begin{bmatrix} cosq_6 & 0 & -sinq_6 & 0 \\ sinq_6 & 0 & -cosq_6 & 0 \\ 0 & -1 & 0 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

and a purely horizontal translation to get from the spherical wrist to the end effector

$$^{6}\mathbf{A_{ee}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_ee \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

To find the transformation matrix that that relates the origin to the location of the end effector, we can multiply the transformation matrices as follows:

$$^{0}\mathbf{T_{ee}} = {}^{0}\mathbf{A_1}{}^{1}\mathbf{A_2}{}^{2}\mathbf{A_3}{}^{3}\mathbf{A_4}{}^{4}\mathbf{A_5}{}^{5}\mathbf{A_6}{}^{6}\mathbf{A_{ee}} \tag{2.2}$$

where the matrix is comprised of:

$$^{6}\mathbf{A_{ee}} = \begin{bmatrix} {}^{0}\mathbf{R_{ee}} & {}^{0}\mathbf{r_{ee}} \\ 0 & 1 \end{bmatrix}.$$

where R^0_{ee} is:

$$\mathbf{R^0_{ee}} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}.$$

These equations were then used in matlab to create a simulation of the model to be used for verification and which would return the end effector pose. The translational component is simply the vector $^{0}\mathbf{r_{ee}}$. However, the rotational component is unintuitive when extracted as a rotation matrix. Therefore, it was converted to euler angles using:

$$\mathbf{\Theta^0_{ee}} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(R_{32}, R_{33}) \\ atan2(-R_{31}, \sqrt{R_{33}^2 + R_{33}^2}) \\ atan2(R_{21}, R_{11}) \end{bmatrix}$$

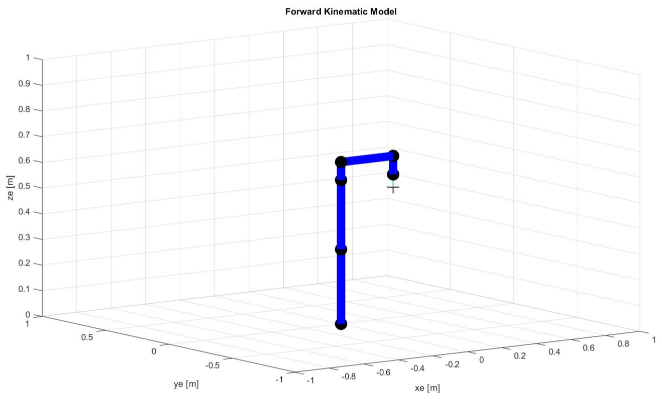

Figure 2: Forward kinematic model simulation

which was then verfied using RobotStudio to ensure the simulated pose of the end effector was consistent with the robot controllers software

2.4 Inverse Kinematics

Solving the Forward kinematic problem to find the joint angles required for a certain configuration is useful. However, tasks are performed in the cartesian space. Thus, a method is required to prescribe a end effector pose in the task space which will return the joint angles required in joint space. This problem is known as inverse kinematics.

A method to solve the inverse kinematic problem numerically requires the use of the Jacobian.

The Jacobian is defined as:

$$\mathbf{J} = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

where J_v and J_ω relate to the linear and angular velocity of the joint in cartesian space to the rate of change of it's joint angles in joint space:

$$\mathbf{v_n^0} = \mathbf{J_v}(q)\dot{q}$$
$$\omega_\mathbf{n}^\mathbf{0} = \mathbf{J}_\omega(q)\dot{q}$$

The Jacobian of the ABB was computed using:

$$\mathbf{J} = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} = \begin{bmatrix} R_{i-1}^0 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} (r_n^0 - d_{i-1}^0) \\ R_{i-1}^0 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \end{bmatrix}$$

The Jacobian for the ABB is then constructed:

$$\mathbf{J} = \begin{bmatrix} J_{v_1} & J_{v_2} & J_{v_3} & J_{v_4} & J_{v_5} & J_{v_6} \\ J_{\omega_1} & J_{\omega_2} & J_{\omega_3} & J_{\omega_4} & J_{\omega_5} & J_{\omega_6} \end{bmatrix}$$

Thus, the geometric jacobian has been constructed. However, to have the ability to prescribe rates of change of angular velocity in roll, pitch and yaw rates, an analytical jacobian is required where:

$$\mathbf{J_A} = \mathbf{T_A}(\mathbf{\Theta_n^0})\mathbf{J}$$

$$\mathbf{T_A}(\mathbf{\Theta_n^0}) = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{E^{-1}}(\mathbf{\Theta_n^0}) \end{bmatrix}$$

$$\mathbf{E}(\mathbf{\Theta_n^0}) = \begin{bmatrix} 1 & 0 & -sin\theta \\ 0 & cos\phi & -sin\phi cos\theta \\ 0 & -sin\phi & cos\phi cos\theta \end{bmatrix}$$

This anaytical jacobian can then be used to reach a desired pose x_d from current pose x_e where:

$$\mathbf{x_e} = \begin{bmatrix} r_n^0 \\ \Theta_n^0 \end{bmatrix} \qquad \mathbf{x_d} = \begin{bmatrix} r_n^0 \\ \Theta_n^0 \end{bmatrix}$$

can be used to form an error function e

$$\mathbf{e} = \mathbf{x_d} - \mathbf{x_e}$$

Taking the derivate gives:

$$\dot{\mathbf{e}} = \dot{\mathbf{x_d}} - \dot{\mathbf{x_e}} = \dot{\mathbf{x_d}} - \mathbf{J_A}(\mathbf{q})\dot{\mathbf{q}}$$

an

Leads to the following control structure:

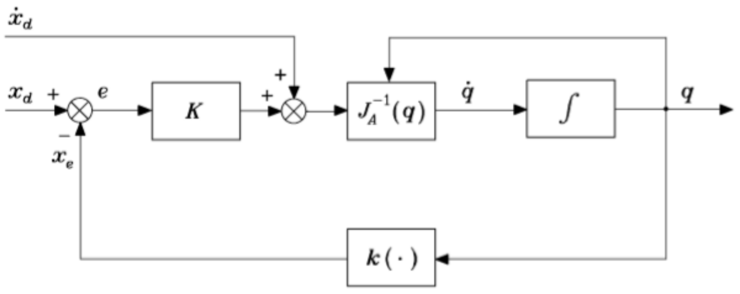

Figure 3: Inverse kinematic model Control Scheme

Which was simulated in Matlab and was used to follow a desired path, as shown:

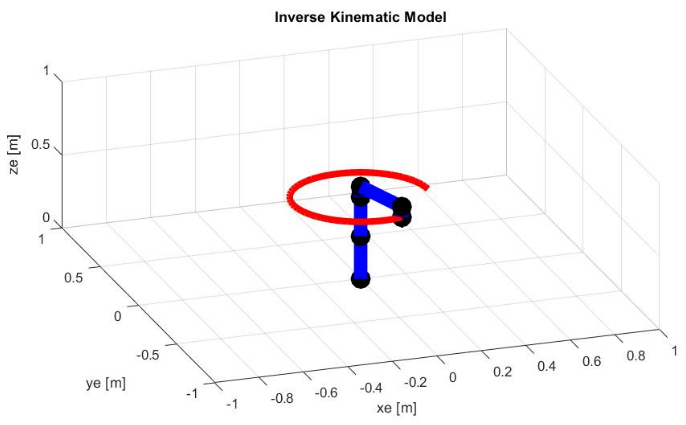

Figure 4: Inverse kinematic model Simulation

Thus, a simulation model of the ABB IRB 120 has been constructed which can follow a desired trajectory. The next section will now discuss the how the robot will be controlled to follow a trajectory such that it picks and places desired objects.

3 Object Classification

What camera was used, how it was set up, how the training data was gathered, labelled and the accuracy validated

3.1 Method

- how the learning actually works in a faster rcnn, with the training set and the validation set used to minimise an error function which backpropogates through the layers, changing their weights

3.2 Dataset Generation

- initially the chameleon cmln-13s2c was to be used. However, specification were not suitable for the task.
- another camera will be used for the second half, in the meanwhile a rasberry pi camera is used - took 200 images of each class in different environments with varying lighting conditions and backgrounds to ensure robustness
- also had negatives, which obscured the image in the form of glare and obstructed views - used an open source software called LabelImg

3.3 Training and Validation

- training was performed on both matlab and tensorflow using inbuilt libraries
- comparison between faster rcnn vs ssd on this dataset
- faster rcnn was more accurate albeit slower, but that is fine as the robot will not be moving very fast. thus faster rcnn will be used in part B

4 Object Localisation

How to perform image transforms and how to localise objects as a pose relative to the base of the robot or the end effector

4.1 Image Jacobians

- how an image jacobian works i.e. maps the location of a point on an image as pixels to the location of the point in cartesian co-ordinates
- we know the end effector and the camera and thus we can get the location of points of interest - The jacobian and transformation for our system to get from camera to certain end points.

The desired points will be found using edge detection. This will produce unique quadrilaterals whose intersection points will be used as desired feature points.

4.2 Simulation

- This can shown using this example image, which is then transformed to give the desired feature points

5 Trajectory Generation

Talk about travelling salesman problem and how it got solved and what constraints had to be used and how i fixed it and had it work for an arbritrary number of objects to be moved max 10

5.1 travelling salesman problem

- topological problem, can use a minimiser to find the optimal path - how to represent it mathematically and the constraints - how this was modified to use in our problem

5.2 Simulation

- examples of how the simulation worked which was used for validation and shows the desired trajectory

6 Visual Servoing

- this task involves using camera input and a desired trajectory to control the serial manipulator

- uses object classifier to classify each object
- This is used by the object localiser to determine its feature points and cartesian co-ordinates
- This is used by the trajectory generator to determine the optimal path
- follows the desired trajectory by controlling the robot arm using visual servoing.

7 Future Work

What still needs to be done to get a hardware demonstration:

- demonstrate visual servoing in simulation
- Configure RobotStudio and the ABB robot controller to receive serial commands from Matlab

8 Conclusion

9 References and Citations

References