Australian National University

Research School of Engineering

College of Engineering and Computer Science

STUDENT ID NUMBER: u6434979

COURSE NAME: INDIVIDUAL RESEARCH THESIS

COURSE CODE: ENGN4200

DUE DATE: **June-07, 2019**

Submission of this assignment constitutes a declaration that:

- No part of this work has been copied from any other person's work except where due acknowledgement is made in the text; and
- No part of this work has been written by any other person except where such collaboration has been authorised by the course lecturer concerned; and
- No part of this work has been submitted for assessment in another course in this or another part of the University except where authorized by the lecturer(s) concerned.

Date: June-07, 2019

Application of Stereo Vision for closed-loop reaching control of a Robotic Manipulator: A Control Lyapunov function learning approach

Student – **Nimish Magre**

U6434979

Primary Supervisor – **Professor Robert Mahony**

Secondary Supervisor – **Zheyu Zhuang**

A thesis submitted in part fulfilment of the degree of
Bachelor of Engineering
Department of Engineering
Australian National University

Acknowledgments:

The research work documented in this thesis would absolutely not have been possible without the constant support I received from both my primary and my secondary supervisors.

I would like to express my gratitude to Professor Robert Mahony and to Zheyu Zhuang for sharing their immense knowledge in the field of robotics and providing productive feedback for during the course of this project on a personal level.

Abstract:

A long-term goal in developing robotic system to aid humans in daily lives is to develop their abilities to act autonomously and in uncontrolled environments. The current robotic systems are able to perform exceptionally well in numerous manipulation tasks and are used extensively in various industries. However, all these tasks performed by these robotic systems involve specifically structured environments. These systems fail to perform even the most mundane manipulation tasks such as reaching and grasping an object in novel uncontrolled environments.

This research therefore aims to contribute towards solving this problem by utilising stereo vision sensors for closed loop reaching control of a 6-DoF robotic manipulator. Specifically, the research aims to use the visual sensor data to train a Convolutional Neural Network in order to regress a Control Lyapunov function and use its partial derivatives with respect to the joint angular velocities of the manipulator to servo-control the manipulator. The goal is to then enable the manipulator to autonomously carry out a multi-target reaching and grasping task using the stereo sensor feed for servo control in real time.

The project develops on previous research by Z. Zheyu et al. [1] which introduced the concept of exploiting the control Lyapunov function to design a velocity controller for the manipulator. This project utilised data from a monocular vision sensor and was able to achieve a successful grasping rate of 78.3 ± 5.4% [1] when tested in the lab environment with multiple coloured cubes as the targets. An aim of the research is also to exploit the extra information of the scene obtained from the stereo vision sensors to verify whether the performance of the network improves in comparison to the monocular vision-based network.

After completing the training process for the stereo vision-based network, it was observed that the network failed to train well enough as the validation loss observed was relatively high and experienced fluctuations between different batches of the sample dataset.

CONTENTS

List of figures:

Chapter 1: Introduction:

1.1 RESEARCH TOPIC:

Application of Stereo Vision for closed-loop reaching control of a Robotic Manipulator: A Control Lyapunov function learning approach

1.2 PROJECT CONTEXT:

In order for a robotic structure to interact with the physical elements in its surrounding, the process of manipulating the robot pose from its natural state to reach the physical element plays a crucial role. Traditionally, this process of manipulating the robot pose has either been carried out by direct human control or through automation under controlled environments. However, these traditional robotic systems fail to perform satisfactorily in unstructured environments without direct human control [2].

A fundamental area of research in the field of robotics focused on solving this issue involves research on autonomous robotic manipulation tasks such as reaching and grasping objects in uncontrolled environments using visual servo control which is also the core research field for this project.

The research aims to train a Convolutional Neural Network (CNN) to effectively learn a control Lyapunov function (cLf) for a closed loop multi-target (colour coded cubes) reaching task {*Figure 3*}. The differential of this cLf with respect to the joint angular velocities of a Universal Robot UR5 manipulator with 6 degrees of freedom are then used in order to servo-control the manipulator in real-time. The CNN has been used specifically to map visual data obtained from a 2nd person stereo-camera feed to the control Lyapunov function and its derivatives with respect to the manipulator's joint angular velocities in order to allow the manipulator to autonomously reach and grasp a coloured cube target.

The project builds on previous research by Z. Zheyu et al. [1] which presented the specific method of exploiting a control Lyapunov function by training a CNN to servo-control the manipulator and utilised a monocular vision sensor for visual perception.

1.3 PROJECT RELEVANCE:

In today's world, extensive use of robotic systems can be noticed in a variety of industrial sectors from auto-assembly lines to assisted care facilities in nursing homes. These systems are able to physically interact with objects and perform a variety of manipulation tasks autonomously within carefully controlled settings. However, even though these systems are able to perform manipulation tasks with tremendous efficiency in controlled environments, most of these systems would not be able to perform the most mundane of human tasks such as picking up and placing an object from one place to another under environments that are not carefully structured [2]. Therefore, research on the autonomous manipulation of robotic systems under unstructured environments holds great value for productive use of these robotic systems in our daily lives.

Recent research work related to solving this issue makes extensive use of visual sensors to guide the robotic systems in novel environments with precision. The use of visual information to guide robotic mechanisms is considerably widespread and is used extensively with manufacturing operations, cameras to track missiles and teleoperation tasks [3]. This project particularly focuses on exploring the possibility of using feedback from visual sensors to help a robotic manipulator in performing a multiple target reaching and grasping task autonomously.

1.4 BRIEF METHODOLOGY:

The following sets of tasks briefly describe the scope of the project research and steps carried out to complete the project:

Table 1: list of goals for project completion

Sr. No.	**Milestone**
1	Learn direct manipulator control (ground truth) - TCP configuration - Pose configuration - Troubleshooting - Emergency stop
2	Experimental Study of monocular vision-based network [1]
4	New Simulator for 1st person - Setup sims using Gazebo platform - Camera placement methods - Collecting experimental data from simulation

5	Teaching Control (proof of concept) - Train and optimise the network to learn effective control parameters
6	Build 1st person Rig - Real world camera placement to control manipulator
7	Experimental Reaching (grasping) - Conduct experimentation to test real world performance

By the end of the research period the first 4 milestones were completed and the network was trained as an initial proof of concept. However, initial results showed that the network failed to learn the control parameters effectively.

Chapter 2: Literature Review:

Vision sensors have long been used with a variety of robotic systems utilised in various industrial sectors for applications such as assembly level part inspection or robot teleoperation [4]. However, these robotic systems have had minimal impact for applications in the real-world within uncontrolled environments. Researchers in the field of robotics have therefore been working towards optimising the ability of these robotic structures to function in uncontrolled environments without human intervention and have observed the lack of sensor capabilities in the earlier robotic systems as a primary cause of this inadequacy [5].

One of the key applications of achieving such a goal would be the category of reaching tasks wherein the robotic manipulator is required to be moved from its current pose to a certain target pose that is defined with respect to a pre-observed environment [2] in order to physically interact with objects. For autonomous manipulation in such reaching tasks, it becomes crucial to estimate the target pose of the manipulator with respect to the surrounding observed environment using sensor inputs.

Early work by Y. SHIRAI et al. [6] involving feedback from a commercial vidicon TV camera to correct the pose of a 7-DoF robotic manipulator for the task of grasping a prism block and placing it in a square box along with research on application of computer vision in the industrial inspection and assembly systems in the early 1980s [7] demonstrated the effectiveness of utilising visual sensor inputs for effective robotic manipulation. Traditionally, these visual sensors were used for manipulation in an open-loop manner wherein the accuracy of the

manipulation depended directly on the vision sensor accuracy and the accuracy of the manipulator control [5]. An open-loop manner of manipulation suggested that the tasks of image processing for the visual data and applying manipulator control were performed independently. Image processing times in such systems are relatively long at around 0.1 – 1 second [4].

To avoid the high dependency on visual sensor accuracy and accuracy of the manipulator control, an alternative to the traditional open-loop robotic manipulation was developed to incorporate feedback from visual sensors to control the pose of the manipulator's end-effector corresponding to a target pose. This alternative process was coined as '*visual servoing'* and can be described as a combination of several crucial fields including kinematics, image-processing, control theory and dynamics [5].

Modern visual servoing systems utilise information obtained from the vision sensors for real-time control as opposed to the earlier methods of determining the representation of structures in the scene initially and then planning the manipulator motion (open-loop control: 'looking' and 'moving') [3]. To utilise the process of visual servo control for a manipulation task in real-time, the two basic philosophies have been Position-Based Visual Servo (PBVS) and Image-Based Visual Servo (IBVS).

PBVS has been the earlier approach and utilises image features to explicitly estimate the target pose relative to the camera. This estimated pose is then used to generate an error signal (difference between the estimated target pose and the current manipulator pose) in the Cartesian space to servo-control the pose of the manipulator. In order to interpret the target pose relative to the camera, this approach utilises the known geometric model of the camera in conjunction with features extracted from the image obtained using the visual sensors. [8].

Early work following this approach for robotic reaching tasks utilised monocular vision data from a single camera mounted directly to the end-effector of the robot. The object pose relative to the camera was then obtain using known feature points from the object in the image plane and an extended Kalman filter was applied to attain a recurring solution for the image-based photogrammetric equations which provided the estimated target pose [9] [10]. To further improve the reliability of this approach in the interaction of the robots with formerly unknown objects, later work by M. Luis et al. [11] and M. Antonio et al. [12] began to focus on estimating the grasp reliability from a number of possible grasps by using probabilistic models to learn from previous interactions of the manipulator with known target objects. These

methods successfully reduced the uncertainty with grasping unknown objects by predicting the performance of a particular grasping option only by analysing visual features for the grasp. Some of the more recent works using the PBVS philosophy by K. Alexander et al. [13] in 2014 and Eric Brachmann et al. [14] in 2015 have also incorporated depth sensors along with the standard RGB images to obtain the depth information of the captured scene and improve the grasping performance when the objects are occluded, present in a large variety or present in a disordered environment.

A particular advantage in using the PBVS approach is the fact that it allows the reaching task to be modelled in terms of a Cartesian pose used widely in the field of robotics [5]; however since the control parameters depend on the estimated target pose, it can be said that the camera calibration and geometric models of the target significantly affect the performance of this approach.

As opposed to the PBVS approach, the Image Based Visual Servo (IBVS) approach computes the control values for the manipulator directly on the basis of image feature parameters. In the context of computer vision, any structural parameter such as edges or corners that can be extracted from an image is referred to as an image feature [5]. By computing the control commands using image features directly, this approach does not require known geometric models for objects and eliminates the errors observed due to slight inaccuracies in camera calibration [3].

Traditional implementations of this approach have the camera attached to the end-effector of the manipulator and use the image jacobian matrix that defines a relationship between the image-feature velocities and camera velocities to obtain the desired camera velocities to move the robot end-effector to the desired target pose without explicitly calculating the target pose [8].

Many of the modern algorithms following this approach have utilised deep neural networks to learn the end-to-end visuo-motor control policies in simulation and have then transferred the learning to perform real-world reaching tasks. J. Stephen et al. [15] computed robot trajectories for a cube grasping task in simulation and trained a Convolutional Neural Network to map the observed images to velocities using domain randomization and were successfully able to perform the cube grasping task in real world with unique environments and varying lighting conditions. Similarly, Z. Fangyi et al. [16] utilised deep CNNs to learn and transfer visuo-motor policies from simulation to real-world using monocular RGB images to train the network

in simulation. They were able to achieve a 93.3% successful grasp rate in real-world experimentation and therefore provided sufficient evidence on the robustness of using CNNs to learn visuo-motor policies even with simulated dataset for robust grasping performance in the uncontrolled real-world environment.

Both the categories for visual servoing discussed above allow for extensive use of stereo vision sensor feed to be exploited for improved accuracy in performing robotic manipulation tasks; for example recent work by S. Pablo Ramon et al. [17], used stereo vision sensors for an autonomous grasping of known objects using Unmanned Aerial Vehicles (UAVs) in real time. The algorithm utilised the stereo vision feed to learn object feature models offline and utilised these models to detect targets and estimate their position online. The research found that the feature-based model was robust to occlusions and outliers.

Chapter 3: Background and Problem Formulation

3.1 HIGH-LEVEL BACKGROUND THEORY

The aim of the research is to utilise information from a stereo vision sensor to develop the control of a Universal Robot UR-5 manipulator with 6 DoF in order for it to autonomously reach and grasp a target from its original position. To do so, a Convolution Neural Network has been utilised to learn a Control Lyapunov function which has then been used to obtain the servo-control parameters for the manipulator to perform the multiple target reaching task.

1. **Convolutional Neural Networks:**

   Computer vision is one of the fastest growing fields in the Artificial Intelligence paradigm and aims to enable robotic machines to understand and view its surroundings similar to humans. A Convolutional Neural Network is a particular algorithm that has contributed significantly to advancements of the Computer Vision field with deep learning and is used extensively for image classification tasks.

   Typical architecture for a CNN used for classification tasks includes some convolutional and subsampling layers, a bias component and optional fully connected layers. The convolutional layers contain certain filters (kernels) with assigned weights, each of which are convolved over the image to produce feature maps. Each feature map is then subsampled using a pooling technique and a bias is added to each feature map [18].

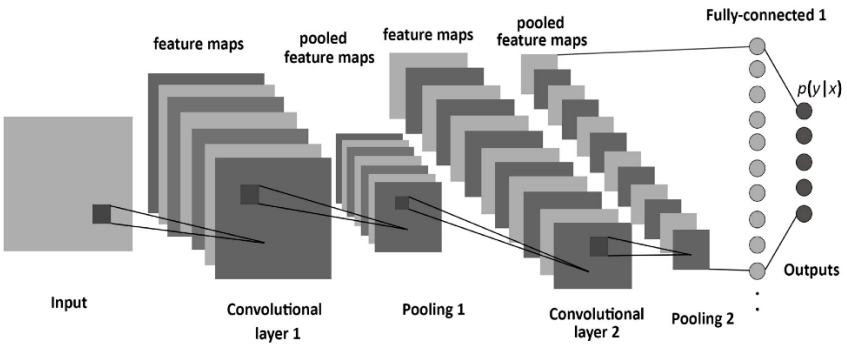

Figure 1: [19] The figure depicts a typical CNN architecture for a classification task consisting of convolutional layers, pooling or subsampling layers and a fully connected layer. Each convolutional layer is obtained by convolving a filter (kernel) with assigned weights over the input. The pooling layers are used to perform downsampling of the convolved features and the full-connected layer is used to obtain the final output or scores for each possible class. [19]

When training a CNN, the input data is passed through each layer of the architecture to obtain the output and this process is known as forward propagation. The output result is then compared to the expected result and a loss function based on this comparison is defined. This loss function is backpropagated through each layer of the network by calculating the gradient of the loss function with respect to each parameter of each layer and based on these gradients, the network weight parameters in each layer are updated. Some of the well-known CNN architectures used commonly in the image classification field include LeNet, ResNet, ALexNet, etc [20].

To gain a basic understanding in building, training and testing CNN architectures, a simple image-classifier network was trained and tested using the PyTorch platform. To train the classifier, the CIFAR-10 image dataset which includes images with 3 channels (RGB), a size of 32 x 32 pixels and 10 classes was used [21]. Testing the network predictions for each class also helped in understanding if using a CNN leads to better classification compared to pure chance-based classification. The following standard order of tasks were performed to develop, train and test the classifier.

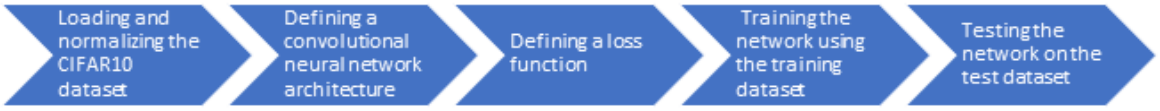

Figure 2: list of tasks followed to train and test a CIFAR-10 image classifier CNN

1. Loading and normalizing CIFAR-10 dataset: The 'torchvision' package, used explicitly with image based neural networks contains popular image datasets such as CIFAR-10 and Imagenet as well as data transformers for images. For use on the Pytorch platform, the datasets were initially transformed into tensors and then normalised in the range [-1, 1]. Normalising the dataset in this range is essential to stabilise the process of backpropagation otherwise the network would overcompensate in a certain weight dimension and undercompensate in the other which is undesirable.
2. Defining the CNN architecture: Following are the layers used for the network architecture:
    - convolution layer 1: kernel size of 5
    - max pooling layer 1
    - convolution layer 2: kernel size of 5
    - max pooling layer 2
    - final fully connected layer: output of 10 neurons

    The Rectified Linear Unit (ReLU) activation function was used throughout the architecture. Eventually, a softmax function which normalises the 10 output neurons from the last FC layer into a probability distribution with 10 probabilities (1 for each class) was used to obtain the image score for each class.
3. Defining a loss function: Since the prediction or classification of the image calculated a probability between 0-1, a cross entropy loss function was used. The logarithmic function leads to rapidly increasing loss values for greater differences between predicted and actual probabilities and leads to relatively small loss value for smaller difference in predicted and actual probabilities. Along with this function, the Stochastic Gradient Descent function with a constant step size (learning rate) of 0.001 was used for weight optimisation.
4. Training the network using the training dataset: The entire input training dataset was fed into and passed through the network twice in mini-batches of 2000 samples.
5. Testing the network on the test dataset: the network was given an input of 10000 test images to predict the classes and the performance of the network was calculated by comparing each image's class prediction to the ground truth.

    The network achieved a success rate (correct class prediction rate) of ~59% which is clearly greater than the 10% success rate that would have been achieved with pure chance-based predictions.

2. Control Theory and Lyapunov Function:

Control theory is an area of mathematics that deals with the control parameters of dynamical processes of engineering machine systems. It aims towards optimising the control parameters of the dynamic systems such that they obtain control stability. To do so, a controller that monitors the difference between the expected outcome **(set-point)** of the dynamic system and the actual outcome is employed and this error signal is used as feedback to stabilise the system in closed loop regulation tasks [22].

Similarly, the desired pose for a robotic manipulator to grasp a particular target object can be considered to be a set-point regulation task. The autonomous grasping system can therefore be defined to be globally exponentially stable in equilibrium at the desired target pose. Based on this categorization of the system at the desired target pose, a scalar continuously differentiable Lyapunov function which decreases exponentially along the desired target poses can be implemented to prove global asymptotic stability of the system at the target pose [23].

3.2 FORMULATION OF CONTROL LYAPUNOV FUNCTION AND MANIPULATOR VELOCITY CONTROL PARAMETERS:

During this research, a CNN architecture was used to learn the value of a compound control Lyapunov function and its differentials with respect to the joint angular velocities which were then used to servo control the manipulator for the multiple target (coloured cubes) reaching task. The six functions correlated to the motion of the 6 manipulator joints are highly non-linear and therefore a CNN is used to approximate the Lyapunov function value directly from the stereo image data instead of trying to learn the non-linear motion attributes.

Since the research involves a reaching task with multiple targets, a compound Control Lyapunov function (cLf) with a minimum at each of the possible target poses is introduced and the control of the system is designed such that the cLf value is minimised [1]. Due to this technique, the conventional approach of first selecting a target and then manipulating the control parameters to reach that target will be avoided and will allow the system to function in an autonomous manner. Therefore, if the target conditions such as their pose or number of targets are changed during the reaching task, the network will always work towards minimising the cLf value and will therefore be robust to such changes.

Following is an explanation of all the elements required to form the control Lyapunov function which is then used to design the velocity controller of the manipulator. Since the research builds on the cLf learning methodology utilised in the monocular vision-based research [1] and because the formulation of the cLf was out of this research's scope, the following equations have been reframed and used from the monocular vision-based research [1]

1. **Pose representation:**

   Since the research makes use of the UR-5 robotic manipulator, its forward kinematics model can be represented as a function of the 6 joint angles $\theta \in \mathbb{R}^{6 \times 1}$ [24].

   The target and the end-effector of the manipulator, each have 6 degrees of freedom and their poses can be represented by elements of the Special Euclidian Group {Se(3)} wherein the pose of a given rigid body frame {A} with respect to a world or reference frame {B} can be denoted using a homogenous transformation matrix as

$$^{B}X_{A} = \begin{bmatrix} ^{B}R_{A} & ^{B}\xi_{A} \\ 0 & 1 \end{bmatrix} \in SE(3) \quad (1)$$

   where ξ represents the position vector of the origin of {A} in reference frame {B} and R represents the rotation matrix to transform the elements of vectors in {A} to elements of vectors in {B} [25].

   If the world frame, end-effector frame and the frame of the target are denoted by {W}, {H} and {G} respectively, then based on the Special Euclidian Space representation of object poses, the pose of the target frame {G} can be denoted in terms of the end-effector frame {H} as:

$$^{H}X_{G} = \begin{bmatrix} ^{H}R_{G} & ^{H}\xi_{G} \\ 0 & 1 \end{bmatrix} \in SE(3) \quad (2)$$

   The absolute pose (in world-frame) of the end-effector can then be represented as a function of the 6 joint angles of the manipulator based on its forward kinematics as $^{W}X_{H} = {}^{W}X_{H}(\theta)$ with $\theta \in \mathbb{R}^{6 \times 1}$ [24] and therefore the relative pose of the target frame {G} with respect to the end-effector frame {H} ca be represented as:

$$^{H}X_{G}(\theta) = {}^{W}X_{H}^{-1}(\theta)\ {}^{W}X_{G}(\theta) \quad (3)$$

2. **Formulation of the compound Control Lyapunov Function (cLf):**

   A Control Lyapunov Function for cases of system stabilisation is scalar, continuously differentiable and positive definite. For the case of this system, the function is expected

to achieve a value of zero at the coordinates of the manipulator joints at the target pose. Since the reaching task involves multiple goals [$\{G_j\}$ where j = 1, 2, …., n], it can be said that for a particular goal $\{G_j\}$, the system is required to drive the relative pose of the target $\{G_j\}$ with respect to the end-effector frame ${}^{H}X_G(\theta)$ to the identity transformation. Based on this information, a cLf for the goal is defined to be

$$\mathcal{V}_j = \left\| {}^{H}X_{G_j} - 1_4 \right\|_F^2 \text{ for j } = 1, \ldots, n \tag{4}$$

where 1_4 is the $\mathbb{R}^{4x4}$ matrix and $\|\ldots\|_F$ is the Frobenius norm. The Forbenius norm or the Euclidean norm can be defined as the square-root of absolute sum of squares for the elements of the matrix norm of an mxn matrix A [26]:

$$\|A_F\| = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{i\,j}|^2} \tag{5}$$

Since the targets are multiple, a compound cLf is used:

$$\mathcal{V}_j = \min\{\mathcal{V}_1, \ldots, \mathcal{V}_n\} \tag{6}$$

For this compound cLf, its value is considered to reach zero at each of the multiple target poses. Saddle points have been introduced in the cLf, in which a descent criterion allows the system to descend at any one of the target poses.

3. Velocity Controller:

The velocity controller used is:

$\dot{\theta}_i = \mu(t)\partial_{\theta_i}\mathcal{V}$, for i = 1,…,6 where $\partial_{\theta_i}\mathcal{V}$ denotes the vector for partial differentials of the cLf [i.e: $\partial_\theta\mathcal{V} = \left(\partial_{\theta_1}\mathcal{V}, \ldots, \partial_{\theta_6}\mathcal{V}\right) \in \mathbb{R}^{1x6}$] and $\mu(t) > 0$ is a positive time varying constant.

Based on the above equation, we can write

$$\dot{\theta} = -\mu(t)(\partial_\theta\mathcal{V}).\dot{\theta} = \|\partial_\theta\mathcal{V}\|^2 \text{ for } \dot{\theta} = (\dot{\theta}_1, \ldots, \dot{\theta}_6)^T \tag{7}$$

Since the proposed Lyapunov function is of a quadratic nature, the control can lead to large displacements. Hence, the constant $\mu(t)$ is chosen such that the control is not affected for small values of $\mathcal{V}$ but is scaled down for larger values:

$$\mu(t) = \min\left\{\frac{1}{\mathcal{V}(\theta)}, 1\right\} \tag{8}$$

The individual angular velocity terms (∂_θ) are computed using the velocity Jacobian J(θ) of the manipulator as $(\Omega, \mathcal{V}) = J(\theta)\dot{\theta}$ where Ω is the angular rigid body velocity of the end-effector with respect to the world frame (${}^{W}X_{H}(\theta)$) and V is the translational rigid body velocity of the end-effector with respect to the world frame (${}^{W}X_{H}(\theta)$).

3.3 CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE:

This section entails explanation of the overall network architecture to accumulate the set of RGB images from the stereo camera source and the manipulator joint encoders to approximate the cLf value and its derivatives using a regressor network.

1. Network Architecture:

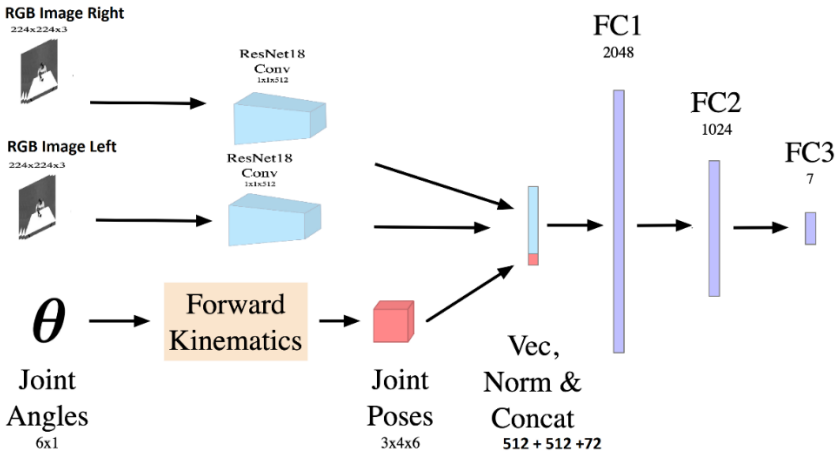

Figure 3: This is the CNN architecture used to regress the cLf value and its 6 DoF partial derivatives. The RGB stereo images are fed through the feature extractor to obtain a 512-dimensional feature vector for each image. The 72-dimensional vectorized joint angle descriptor encodes the 6 joint angle poses. The two 512-dimensinal image features and the 72-dimensional joint angle descriptor are normalized and concatenated and then passed through the fully connected (FC) layers. The 7 outputs from the last layer are mapped to the cLf $\widehat{V}(\theta)$ and its 6 Dof partial derivatives $\partial_\theta \widehat{V}(\theta)$

The overall network architecture is explained through its individual sections below:

Parallel Feature Extractors:

This section of the network architecture has been used to obtain the image feature vectors from the pairs of RGB images collected using a second person stereo camera feed.

The extractor is based on the ResNet18 CNN that was developed initially for image classification tasks [27] and utilises residual blocks which improve the learning process for deep neural networks as will be explained in the section after this. The network architecture for ResNet18 has 1 convolutional layer, 8 residual blocks (explained below) with 2 layers and 1 fully connected layer at the end. For the purpose of this network, the last fully connected layer which gives the results for classifying tasks has been removed to obtain the 512-value output representing the image features. The weights for the ResNet18 network are initialised as its pretrained weights on the ImageNet [28] dataset.

Research conducted recently for obtaining rectified stereo-image feature vectors in order to compute the disparity of each pixel from the two feature vectors by A. Kendall et al. [29] has implemented a similar technique of utilising parallel image feature extractors with shared weights and suggests the use of parallel computation for faster processing and effective weight optimisation. The other option of using separate feature extractors with separate weights for each of the pair of stereo images would certainly imply greater computational cost.

Based on this research, two parallel ResNet18 feature extractors with shared weights are used to initially obtain the two (512-dimensional) feature vectors. These feature vectors are then normalised and concatenated to obtain more information about the scene from the images before finally being concatenated with the 72-dimensional joint descriptors.

ResNet:

In general, deeper neural networks are expected to improve the learning accuracy compared to shallower networks with lesser number of layers. However, as the gradient is backpropagated through a large number of layers, it becomes substantially smaller by the time it reaches the earlier layers which leads to the problem of vanishing

gradients for earlier layers. This problem results in a limit to the improvement in accuracy that the deeper neural networks can achieve [27].

However, when identity mappings are stacked in a deeper neural network to replicate a shallower network, the developers of ResNet argue that the accuracy should not degrade. To solve this issue, the authors of ResNet suggest the use of residual blocks [27] as shown below:

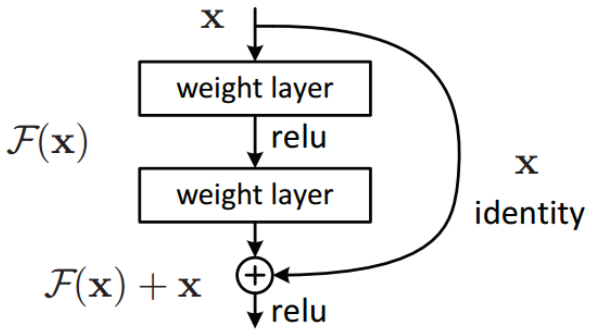

Figure 4: Residual Block: taken from [27]. Consider x as the input to the neural network block and H(x) as the desired output to be learnt. F(x) is the residual between the input and the output (F(x) = H(x) – x). The identity skip connection allows earlier layers in deeper neural networks to receive higher gradients and therefore updates weights for all layers more optimally

Consider the neural network block above to have 'x' as the input and H(x) as the desired output to be learnt. Consider the residual between the input and the output to be $F(x) = H(x) - x$. Rearranging this function, we get $H(x) = F(x) + x$.

Traditional network blocks try to directly learn the output distribution H(x). However, due to the identity skip connection used with residual blocks, it can be seen that the residual block tries to learn the residual function $F(x)$. Therefore, the issue of optimizing the number of layers required for learning and vanishing gradients for earlier layers during backpropagation is solved when the value of the residual function is set to zero. This allows the skip connection to be used during back-propagation and therefore allows the initial layers to receive larger gradients which improves the accuracy for deeper networks.

Joint Angle Decoder:

The forward kinematics model of the manipulator is highly non-linear, therefore, in this network, the forward kinematics elements are explicitly computed at the beginning and

are then transformed into geometric joint feature descriptors. The obtained joint poses for the 6 joints are then vectorized to obtain a 72-dimensional joint descriptor which includes the physical joint positions and orientation vectors as explicit terms.

Regressor:

The three fully connected layers function as regressors to obtain 2084, 1024 and 7 neurons respectively. The 7 neurons obtained from the last layer are then mapped to a single Control Lyapunov Function $\hat{V}(\theta)$ and the 6 DoF partial derivatives $\partial_\theta \hat{V}(\theta)$ to be used for servo control of the manipulator. This network regressor can then be expressed in terms of a function as:

$$h_\psi(\tilde{f}(I), \tilde{g}(\theta)) \tag{9}$$

where ψ represents the weights of the fully connected layers. For each of these FC layers, the Recitifed Linear Unit function has been used for activation.

1. **Siamese Regression:**

For a small batch of 'm' samples, a loss function that can be used to support the learning in the CNN, is the mean square loss function for the cLf and its 6 partial derivatives as shown below:

$$l_d = \frac{\alpha}{m}\sum_{k=1}^{m}\left\|\hat{\mathcal{V}} - \mathcal{V}_k\right\|_F^2 + \frac{\beta}{6m}\sum_{k=1}^{m}\sum_{i=1}^{6}\left\|\widehat{\partial_{\theta i}\mathcal{V}_k} - \partial_{\theta i}\mathcal{V}_k\right\|_F^2 \tag{10}$$

where α and β are positive weighting factors.

Based on the regressor model provided in the section above, it can be seen that the network architecture does not enforce the coupling of the cLf value with its derivatives (no interdependency). In order to obtain robust control parameters through the learnt cLf it is required that

$$\hat{\mathcal{V}}(\theta + \Delta\theta) \approx \hat{\mathcal{V}}(\theta) + \partial_\theta \hat{\mathcal{V}}(\theta).\Delta\theta \tag{11}$$

However, with the suggested loss function 'l_d' in (*eq-10)*, the network will learn the control value $\widehat{\partial_{\theta i}\mathcal{V}}$ through an end-to-end learning architecture without factoring the coupling equation (*eq-11*) which means that learning the cLf will only help with monitoring system performance in real time but will have no effect on the derived differential values.

Therefore, the network uses a Siamese architecture shown in **(Figure 5: Siamese Regressor)** to explicitly enforce the coupling condition from *(eq-11)*.

It can be assumed that for small variations in the joint angles, the images obtain from the stereo vision sensors will not differ substantially. The Siamese network will therefore use the network feature weights Φ to obtain image features along with perturbed joint angles to generate two separate values of the cLf and two separate sets of differentials. Using the perturbed joint angle $\Delta\theta_i \in \mathbb{R}^{6x1}$ for the ith joint angle, its cLf value output of $\hat{\mathcal{V}}(\theta + \Delta\theta_i)$ can be compared with the original cLf value (with unperturbed joint angles) output of $\hat{\mathcal{V}}(\theta)$. This comparison would therefore consider the coupling equation and a new loss function based on this comparison can be written for a small sample batch of m-samples as:

$$l_c = \frac{1}{6m}\sum_{k=1}^{m}\sum_{i=1}^{6}\left\| \widehat{\partial_{\theta_i}\mathcal{V}_k}\,(I_k,\theta_k) - \frac{\hat{\mathcal{V}}(I_k,\theta_k+\Delta\theta_i)-\hat{\mathcal{V}}(I_k,\theta_k)}{\Delta\theta_i} \right\|_F^2 \qquad (12)$$

Combining the direct end-to-end learning loss with the loss function above, the final loss can be written as

$$\mathcal{L} = l_d + \gamma l_c \qquad (13)$$

where γ is a positive weight parameter. The Siamese loss term 'l_c' can act as a regularisation parameter to ensure that the control inputs will tend to decrease the cLf.

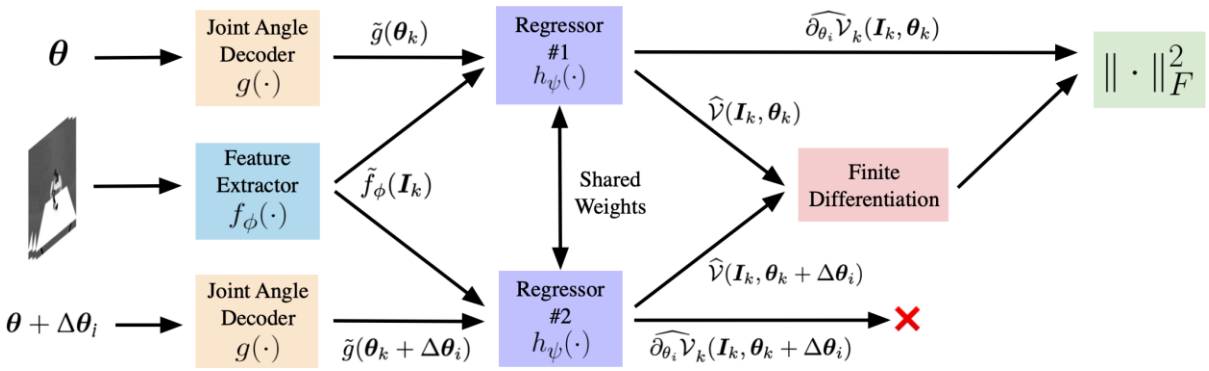

Figure 5: Siamese Regressor: taken from [1]. 2 identical regressors sharing the same weights and stereo-image input features $\tilde{f}_\phi(I_k)$, with original and perturbed joint angle descriptors $\tilde{g}(\theta)$ and $\tilde{g}(\theta+\Delta\theta)$ respectively are fed in. Regressor #1 calculates the original inferred cLf and its partial derivatives whereas Regressor #2 calculates the cLf and its partial derivatives with the perturbed joint angles. The partial derivatives obtained from the regressor using perturbed angles are discarded and the 2 cLf values are used to numerically approximate the partial derivative with respect to θ_i. The contrastive loss for this ith term is calculated as the root-mean squared distance of the difference between the originally inferred value and the numerically approximated value of the partial derivative.

Chapter 4: Use of Gazebo7 simulator

4.1 REQUIREMENT AND JUSTIFICATION TO USE SIMULATION FOR TRAINING THE NETWORK:

Since the research aims to learn the velocity control parameters for the robotic manipulator using a CNN, it is crucial to train the network with a significant amount of dataset {stereo-images and vectorized joint poses for different target poses} to obtain reliable weights for all layers of the network. Relatively small training dataset shows that the network tends to overfit [30] the dataset and may seem like it is performing very well due to low training loss. However, on supplying the network with the testing dataset, the network will tend to perform poorly since its weights have only been optimised for the small training dataset that it overfitted (similar to memorizing) instead of optimising network weights generally.

With the monocular vision-based network [1], it was observed that over 100,000 sets of simulated images were required to satisfactorily train the network. Therefore, generating the stereo-image dataset of a similar size for training the network with real-world robot manipulation experience would be tremendously time consuming. Simulators, however, have the potential to generate the required training datasets while consuming significantly lesser time and effort and can be used as an alternative to obtain the training and validation dataset.

For a similar table-top reaching task with a 7-DoF robotic arm reaching and grasping a blue cuboid in a clutter with a velocity controller, the researchers (Z. Fangyi et al. [16]) used a simulation to learn visuo-motor policies and transferred the learnt policies to the real-world to achieve a success rate of about 93.3% at the end of the learning process. Another work of research by V. Ulirch et al. [31] was successfully able to train a deep CNN using simulated data for a visual robotic grasping based task in simulation and test it. A depth sensor was wrist-mounted on the robot and the CNN was used to learn a function defining the distance to true grasps for image-based graph configurations. Moreover, the trained network was also provided significant robustness to image sensor noise and disruption to the target object when performing the grasping task.

Although, a risk with simulated dataset could be modelling errors (simulated setup does not depict real-world conditions) and therefore the transfer learning would not be effective enough. However, very recent work by A. Marcin et al. [32] suggests the randomisation of simulation during training for effective transfer learning from simulation to real-world. This research was also focused on utilising a manipulator to interact (push) an object and was able to maintain a

strict level of performance despite training the network solely using simulated data. The conclusion from these researches in similar fields along with a successful grasp performance of 78.3 ± 5.4% with the monocular vision-based network formed by Z. Zhuang et al. [1], which was also trained with simulated dataset provides sufficient evidence to use a simulator for training the network.

4.2 SETTING UP THE SIMULATION ENVIRONMENT:

The following image displays the expected real-world setup with the type of manipulator, targets and the poses of the relevant components to be replicated in the simulator:

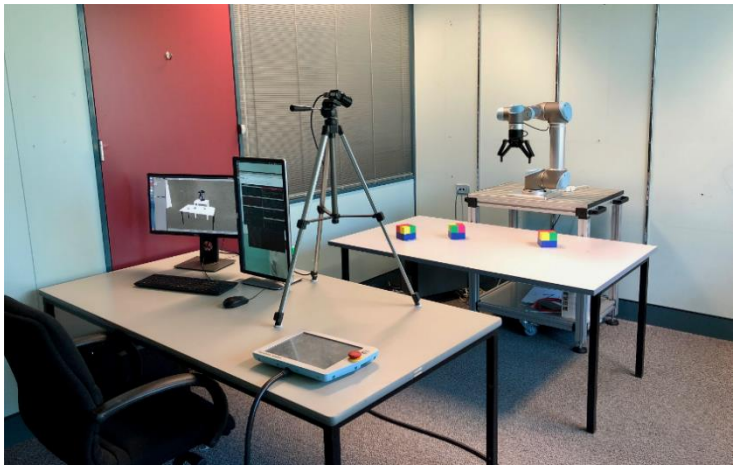

Figure 6: taken from [1]: a picture of the real-world experimentation setup showing the positioning of the camera, the targets (3 colour coded cubes) and the Universal Robot UR5 manipulator along with the Robotic Adaptive 2f-140 gripper attached to the end-effector.

Since the task involves multiple target reaching, 3 coloured cubes have been chosen as the targets for initial experimentation due to relative ease of modelling the cubes.

- The specific manipulator used is the Universal Robot UR5 manipulator with 6 degrees of freedom along with a Robotic Adaptive 2f-140 gripper attached to its end-effector.
- the camera is approximately 1.6m above the ground and 1.7m away from the manipulator base joint
- the camera is inclined towards the manipulator workspace at an angle of 21°

Gazebo7 is a simulation platform that allows to create 3D-dynamic multi-robot environments, is open source and has high fidelity [33] making it a suitable platform to recreate the real-world setup and obtain the training and validation dataset. In order to collect the training and

evaluation dataset for training the network, it is necessary to establish communication with the stereo-camera, the manipulator and the gripper. The ROS platform [34] has been used along with the Gazebo7 simulator for this purpose. Gazebo7 has also been chosen because the ROS control for the manipulator, gripper, etc is steady for the simulator and the hardware.

To establish all the necessary real-world components in the simulator, each component needs to be modelled as a mobile structure in the simulator. Structures in Gazebo are defined in the Universal Robot Description Format (URDF) and have the following general specifications:

- Structure or objects are defined primarily through links and attributes. Links allow the connection of different objects or structures to one another through the definition of a child and a parent link for each object.
- Each link has a name, an inertial attribute, a collision attribute, a visual attribute and is connected to another link through a joint.
- The inertial attribute of a link specifies its physical attributes such as its total mass and mass distribution
- The collision attribute indicates the dimensions and physical attributes of the link whereas the visual attribute defines how the link will be displayed in the simulator (its size, colour, etc). Usually, the 2 attributes have the same parameters.
- The joint to which the link is attached can be defined specifically based on the required movement for the link (ex: fixed or revolute)
- In order to add controllers for a link, certain Transmission parameters describing the relationship between the joint governing the link and the actuator used for motion are defined. One transmission can therefore be defined for each joint.
- The control parameters for a joint can be configured in a separate 'yaml' file and eventually loaded in the launch file to control a joint's state

For the case of this research, the Gazebo7 model representing the real-world setup was already created during previous work on using monocular vision for the reaching task [1] and was modified to replace the monocular vision camera with the Gazebo multicamera plugin to accommodate a stereo camera. As specified earlier, the ROS platform was utilised to gain

access to the synchronised image feed from each lens of the camera. The resolution for the stereo camera in simulation was set to 768 x 1024.

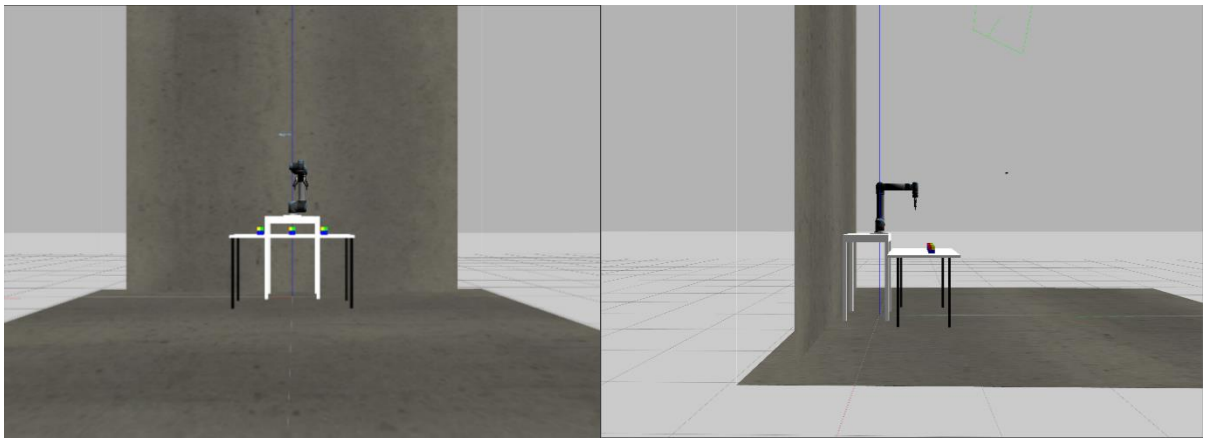

Figure 7: Simulated setup to replicate the real-world experimental setup. The setup consists of the floor, the background wall, the camera, the coloured cube targets, the table on which the targets are placed, a base for the UR5 manipulator and lastly the manipulator itself with a gripper attached to its end-effector. fig a) represents the front view whereas fig b) represents a side-view of the setup

A sample of the synchronised image for the workspace through the stereo-lenses is displayed below:

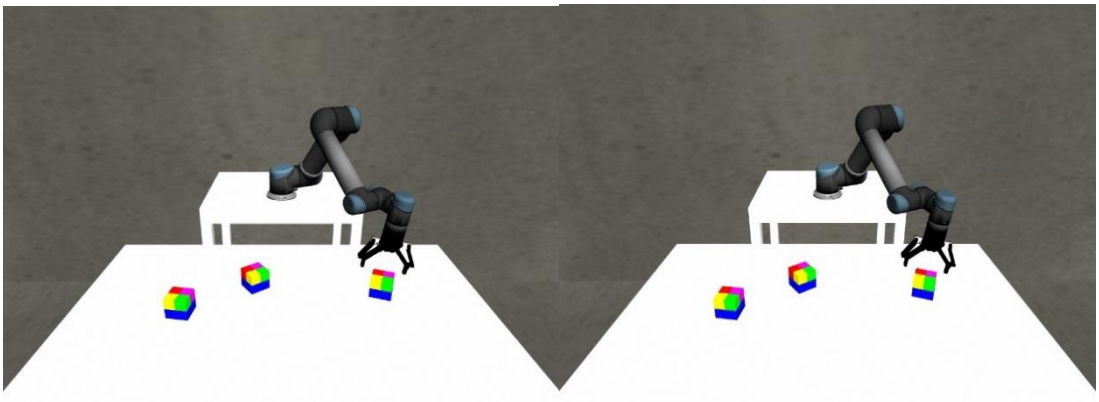

Figure 8: fig a) left stereo lens image of workspace in simulation. Fig b) right stereo lens image of workspace in simulation

Shadow rendering has been turned off in the simulator since multiple light sources are used to depict the real-world setup and Gazebo cannot produce realistic shadows with multiple light sources.

4.3 SAMPLING AND DATA COLLECTION:

This section explains the methodology used to obtain the stereo-image dataset for different target and manipulator poses to be used for the purpose of training and evaluating the network in simulation.

Initially, a large dataset representing different cube (target) poses and joint configurations was generated. When sampling the data, each target cube was modelled to have a 70% probability of appearing in front of the manipulator (i.e the workspace: within 1 x 0.4m in the x and y planes) whereas the target poses with respect to the Z-axis were sampled through a uniform distribution [1].

To sample the end-effector positions, a random target among the three cubes is selected and the end-effector positions are sampled uniformly on semi-spheres whose centres are located at the centroid of the target. For these hemispheres, half have radii sampled through a normal distribution with 0 mean value whereas the radii for the other half are sampled using a uniform distribution with the lower bound set to 0. To make sure that the end-effector is always pointing downwards or towards the targets, the Euler angles of the end-effector frame for every position are sampled on 3 uniform distributions with constrained upper bounds [1].

The collected joint configurations and cube pose quaternions are saved and the data from this file is then used to generate the image-pairs from the stereo camera in the simulator for each corresponding target pose and joint configuration. An important step carried out when collecting the images from each camera is the time synchronisation of the two cameras. Since the stereo-camera is configured to capture 30 images per second, it is important for the training purpose to capture both images at the same time-stamp.

For each of the joint configurations generated above, interpreted values of a compound control Lyapunov function and its 6 partial derivatives are generated as the ground truth based on these configurations. As explained in section {**FORMULATION OF CONTROL LYAPUNOV FUNCTION AND MANIPULATOR VELOCITY**}, for a particular goal $\{G_j\}$ for j =1, 2, …, 6, the target pose in relation to the end-effector of the manipulator $H_{X_{G_j}}(\theta)$ is driven to the identity matrix and the cLf for that goal is defined as:

$$V_j := \left\| H_{X_{G_j}} - 1_4 \right\|_F^2, \text{ for } j = 1, \ldots, n \tag{14}$$

After calculating the cLf with respect to all targets, the minimum of the lot is chosen as the compound cLf value and the partial derivative of this cLf with respect to each of the 6 joint angles is calculated. The calculated compound cLf value and the partial derivatives are used as ground truth when training the network in the later stages.

Based on the data generated in the previous steps, the following classes of data are stored in a hierarchical data format:

1. the generated image-pairs with labels,
2. the joint configurations,
3. The interpreted (ground-truth) values of the compound cLf and its derivatives for each configuration,
4. The absolute vectorized pose of the 6 joints with respect to the world frame for each joint configuration with original and perturbed joint angles (7 x 72-dimensional joint descriptor)

The generation of the image-pairs, joint configurations, the interpreted cLf value and its derivatives values has been explained at the beginning of this section; however, to generate the absolute vectorized pose of the 6 joints with respect to the world frame, it is important to understand the forward kinematics of the manipulator explained as follows:

4.4 FORWARD KINEMATICS OF THE MANIPULATOR

The absolute vectorized joint poses are stored as a matrix made of 7 rows and 72 columns. It represents the vectorized poses of the original joint angles (row 1) and the vectorized poses of the perturbed joint angles (row 2 to 7). In order to obtain the vectorized poses of the joint angles with respect to the world frame, the forward kinematics of the manipulator are expressed using the Denavit-Hartenberg (DH) representation [35] [24].

The D-H convention utilises 4 parameters derived from specific aspects of the geometric relationship between 2 coordinate frames represented by the homogenous transformation matrix 'A' which specifies the position and orientation of a frame attached to the i^{th} link (O_i, X_i, Y_i, Z_i) with respect to a frame attached to the i-1^{th} link $((o_{i-1}, x_{i-1}, y_{i-1}, z_{i-1})$. For a particular homogenous transformation matrix A_i, these 4 parameters can be defined as

1. the link length (a_i): this is the distance between the Z_i and the Z_{i-1} axes measured along the x_i axis.
2. link twist (α_i): this is the euler angle between the Z_i and the Z_{i-1} axes measured by the right-hand rule in a plane normal to the X_i axis
3. link offset (d_i): this is the distance between the origin (O_{i-1} and the intersection between the X_i axis and the Z_{i-1} axes measured along the Z_{i-1} axis
4. joint angle (θ_i): this is the euler angle measured between the X_i and the X_{i-1} axes measured in a plane normal to the Z_{i-1} axes

The homogenous transformation A, can be then calculated as

$$A = \begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\alpha) & \sin(\theta)\sin(\alpha) & a * \cos(\theta) \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & a * \sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (15)$$

This transformation matrix can be described as a function depending on a single variable (θ: joint angle) where the remaining three parameters remain constant for a given link. In order to obtain the position and orientation of the tool frame expressed in terms of the base coordinates, the transformation-matrices are used in the form: $T^0_n = A_1 \cdots A_n$. So if Link 1 is connected to the base frame and link 2 is connected to link 1, then the position and orientation of link 2 with respect to the base frame can be defined using $T^0_2 = A_1 \, . A_2$ where A_2 is the transformation matrix of link 2 with respect to link 1 and A_1 is the transformation matrix of link 1 with respect to the base link.

To obtain the vectorized joint poses for the Universal Robot UR5 6 DoF, the following diagram explains the links and joints used to define the manipulator

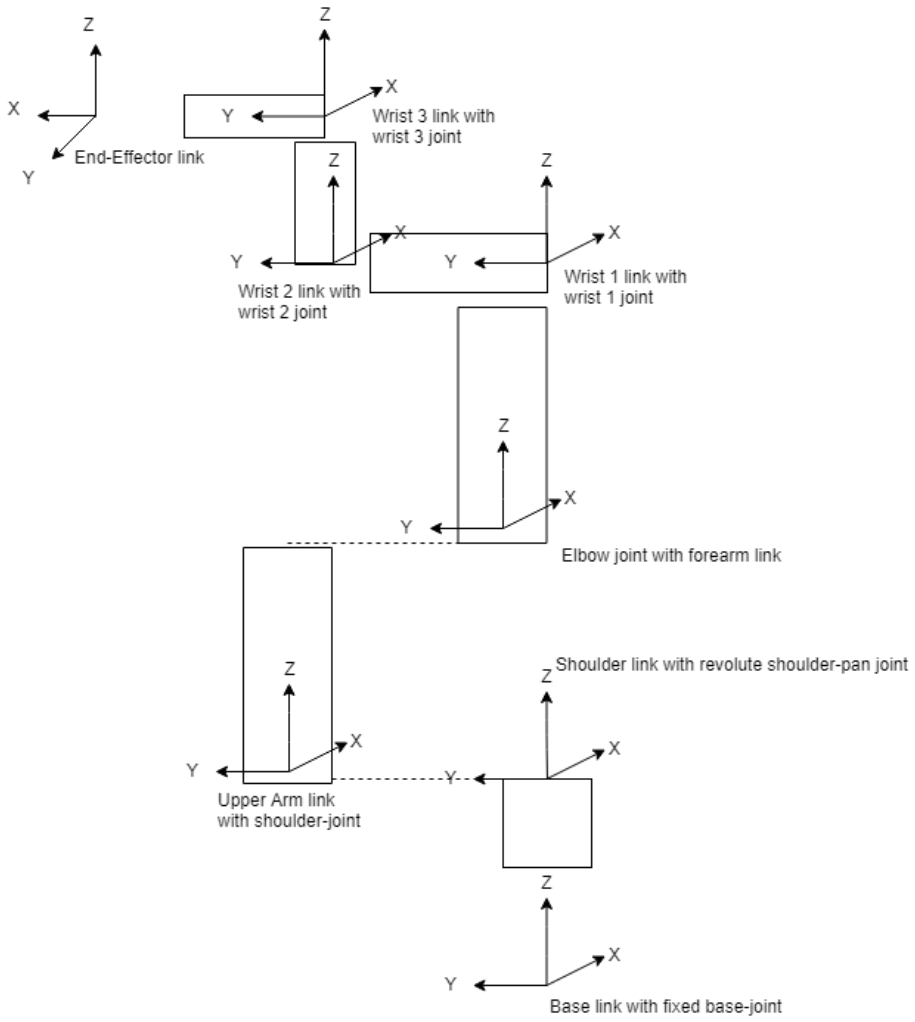

Figure 9: The schematic and frame assignment to determine the vectorized joint poses of the Universal Robot UR5 6-DoF manipulator

The orientation and position of the base joint is initially specified using the 4 x 4 'A' transformation matrix. The joint pose for each of the 6 joints is calculated with respect to the world frame. Using the standard 'd', 'α' and 'a' parameters for each of the 6 joints for the Universal Robot UR5 6 DoF manipulator [24], each joint pose can be described by its (4x4) homogeneous transformation matrix {Ai} with 'i' ranging from 1 to 6.

The last row of every (4x4) joint pose matrix is deleted as that last row values of [0 0 0 1] part of the A-matrix does not provide vital information regarding the pose of the joint. After deleting this row, the remainder of the matrices are converted in a single vector.

Each matrix representing the joint poses was initially (4x4) = 16 elements

After deleting the last row, the new matrix size was (3x4) = 12 elements

After converging to a single vector, each matrix was (1x12) in size and appending each of the 6 matrices together leads to a total size of (1x72) for the 'vectorized_joint_poses' matrix.

Similarly, the 72-dimensional joint pose descriptor after perturbing each of the joint angles is calculated to finally obtain the vectorized joint pose descriptor of size (7 x 72) where the first row represents joint description for unperturbed joint angles.

Chapter 5: Network Training and results:

This section will detail the 3 important steps of forward propagation, backward propagation and optimisation involved in training the regression network:

5.1 FORWARD PROPAGATION:

Forward propagation is the process of propagating the input data through each layer of the defined network architecture and finally arriving at an output to be compared with the desired output or ground truth. The dataset required to efficiently train the network can be divided into the training and the evaluation dataset. For training the CNN in this case, a training dataset with 140,000 samples and an evaluation dataset with 10,000 samples was used. Another hyperparameter used in this process is that of the 'batch size'. It refers to number of samples passed through the network before calculating the error gradients with respect to each layer and optimising the network weights. A mini-batch size of 64 was chosen in this case for the training dataset as relatively smaller batch sizes consume lesser GPU memory and tend to converge the loss function to a more generalised flatter gradient as opposed to large batch sizes [36] [37].

During the forward propagation of the network, the images belonging to the training dataset were pre-processed as follows to achieve data-invariance for varying real-world conditions before being input to the network:

- a padding of 20 was added to the top and bottom parts of the image
- the image was randomly rotated with between -3º and +3º
- the image was randomly rescaled with a scale between 0.95 and 1.05 times the original image, and then resized to obtain an image size of 224 x 224. the image was then cropped from its centre to 224 x 224

- the contrast, hue, saturation and brightness of the image were randomly modified at a maximum of 10%

The 3 inputs to the network are the left and right pre-processed images from the stereo image dataset and the vectorized joint poses. The network obtains the image features (512-dimensional) using the ResNet18 network [27] without its final Fully Connected layer. During the training of the network, the weights for this feature extractor are initialised based on a pre-trained ResNet18 network on the 'ImageNet' dataset [28].

The proposed Siamese regressor {Figure 5}, requires both the perturbed and unperturbed vectorized absolute joint poses to be passed through the network. Instead of these parameters to the network in a sequential fashion, a single forward pass is utilised to obtain the outputs by passing these parameters through the network. To do this, the 7 vectors providing the joint poses for the 1 unperturbed and 6 perturbed joint angle measurements are loaded along the batch dimension and the forward pass is implemented on the GPU to save on computation costs. For all the 6 joints a small angular perturbation value of $\Delta\theta = 0.05$ is utilised to obtain the Siamese loss (*eq-12*).

The output of the network after implementing the forward pass is a single inferred cLf value, 6 partial derivatives of the cLf inferred directly from the regression network and 6 partial derivative values with the perturbed joint angles obtained from the Siamese regressor. These output values are further used in the next section to obtain the training loss of the network and to optimise the network weights based on this loss.

5.2 BACKPROPAGATION AND OPTIMISATION:

In order to backpropagate the error gradients through the network layers, it is crucial to define a loss function that measures the difference between the actual and expected network outputs. The three outputs from this regression network {Figure 3} include the network inferred cLf value, the regression network inferred values of the 6 partial derivatives of the cLf and the 6 numerically approximated partial derivative values obtained from the Siamese regressor {Figure 5}. Based on these outputs, the compound loss function defined in *eq-13* has been used to find the network gradients and has been provided below:

$$compound\ loss\ (\mathcal{L}) = l_d + \gamma l_c$$

The individual loss functions l_d and l_c have been defined in *eq-10* and *eq-12* respectively and are repeated below:

$$l_d = \frac{\alpha}{m}\sum_{k=1}^{m}\left\|\hat{\mathcal{V}} - \mathcal{V}_k\right\|_F^2 + \frac{\beta}{6m}\sum_{k=1}^{m}\sum_{i=1}^{6}\left\|\widehat{\partial_{\theta i}\mathcal{V}_k} - \partial_{\theta i}\mathcal{V}_k\right\|_F^2$$

This function computes the loss between the regression network inferred cLf and partial derivative values and their respective ground truth values. The positive weight parameters α and β have been chosen to be 1 and 6 respectively.

$$l_c = \frac{1}{6m}\sum_{k=1}^{m}\sum_{i=1}^{6}\left\|\widehat{\partial_{\theta i}\mathcal{V}_k}\,(I_k,\theta_k) - \frac{\hat{\mathcal{V}}(I_k,\theta_k+\Delta\theta_i) - \hat{\mathcal{V}}(I_k,\theta_k)}{\Delta\theta_i}\right\|_F^2$$

This is the Siamese loss function and computes the loss between the network inferred partial derivatives and the approximated partial derivatives. The positive weight parameter γ for this function was specified to be 0.5.

Using this compound loss function, the network gradients are calculated, and the Adam optimiser is then used to optimise the network weights. Unlike the classical Stochastic Gradient Descent approach which maintains a constant learning rate for updating all the network weights, the Adam optimiser uses the average first and second moments of the gradients to adapt the parameter learning rates. In the original research paper, the authors experimented with different optimisers on the CNN with the CIFAR-10 image recognition dataset and demonstrated the relatively superior effectiveness of the Adam optimiser over others such as AdaGrad and SGDNesterov [38].

5.2 EXPERIMENTAL OBSERVATIONS AFTER TRAINING THE NETWORK WITH RELATIVELY SMALL DATASET:

When training a CNN with a certain dataset, two very important concepts to understand include the concept of data overfitting and underfitting. In the case of data overfitting, the network models the training data and learns its detail to such an extent that it has a negative impact on the performance of the network with unseen data. Inversely, the network models the training dataset too sparsely when experiencing underfitting. The network hyperparameters therefore need to be optimised throughout the training process to avoid these situations.

The process of initially training the network with a small dataset is utilised to check whether the network overfits the training dataset since it is very small in order to verify correct implementation of the network.

Hence, the network was initially trained with a relatively small training dataset of only 1000 samples, an initial learning rate of $5e^{-4}$ and a learning rate scheduler to reduce the learning rate by a factor of 0.5 after experiencing a plateau on the validation loss for at least 5 epochs was utilised. The following training performance was observed:

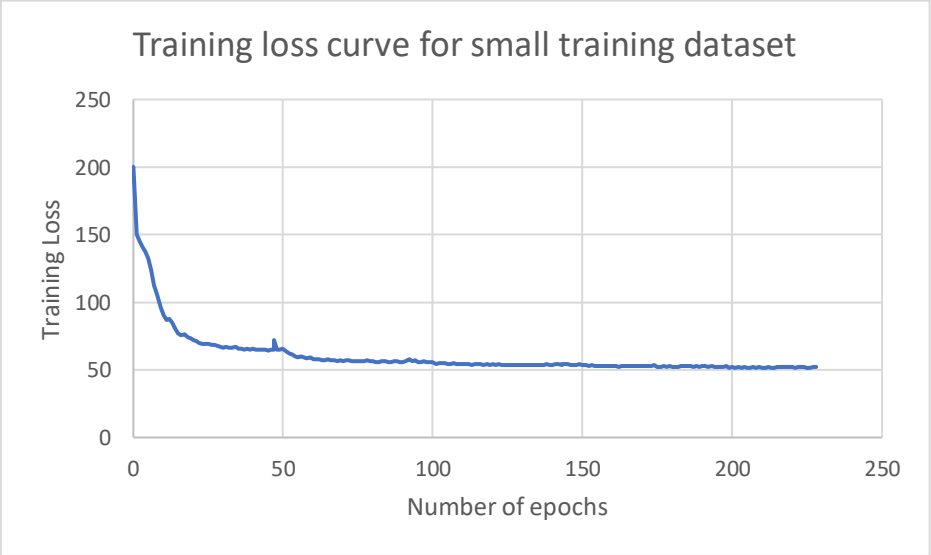

Figure 10: Average training loss curve for network trained using small training dataset of 1000 samples to check if network overfits the dataset

The training loss curve clearly shows that the network was unable to overfit the training dataset as the training loss is stuck at a relatively high value of around 50, suggesting the need to re-evaluate the network implementation.

An initial conjecture was to assume that the parallel feature extractors were not implemented properly and were not able to share weights due to incorrect image samples being passed them; therefore, another experiment using a single feature extractor was implemented with only the right lens image features. To depict the stereo image dataset, the feature obtained for the single right-lens image were copied and concatenated together to utilise an image pair similar to the stereo image pairs but without using 2 parallel feature extractors.

Upon training the network with the above implementation of the feature extractor, a similar result to the previous experiment were observed confirming that the parallel networks were

implemented in the correct manner initially. The network was still unable to overfit the small dataset.

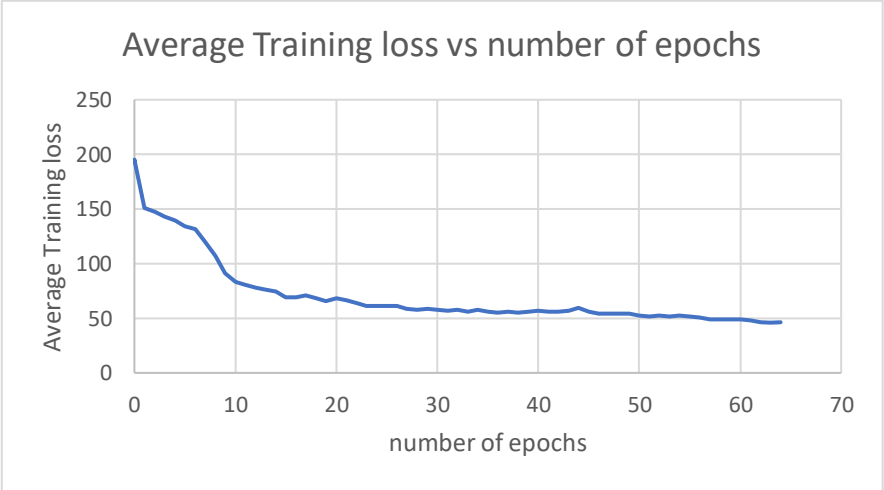

Figure 11: Average training loss curve for network trained after removing one of the feature extractors and using only the right-lens image dataset. The network mimicked stereo image features by copying and concatenating the right-lens image features. This experiment was carried out to verify if the parallel feature extractors were implemented correctly. A small training dataset of 1000 samples was used to check if network overfits the dataset

These results prompted the re-evaluation of the data collection implementation along with the network architecture implementation and certain coding errors were then found and fixed. The network was trained again to verify if the training performance led to data overfitting and positive results were then obtained as shown below:

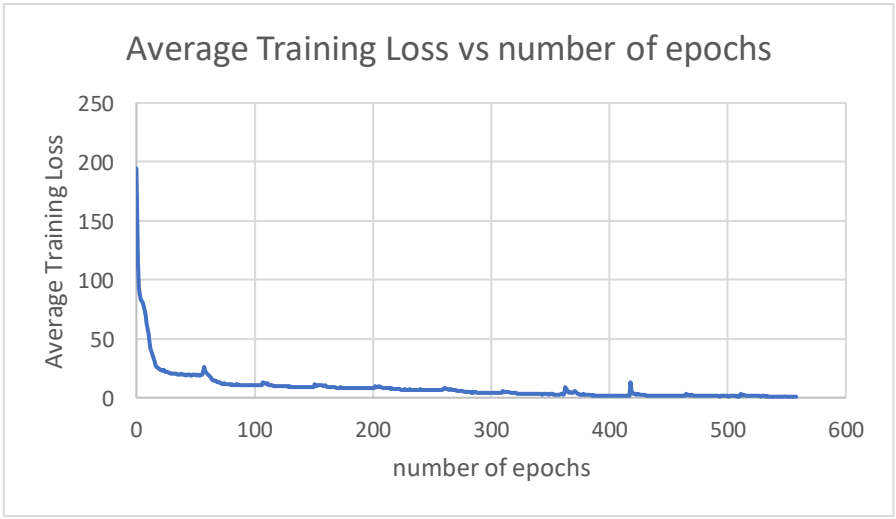

Figure 12: training loss curve for small training dataset of 1000 samples for the network after correcting dataset collection errors

5.3 EXPERIMENTAL OBSERVATIONS OF TRAINING THE NETWORK WITH COMPLETE DATASET:

After obtaining the expected results on the relatively small training dataset, the complete set of 140,000 training samples and 10,000 validation samples was utilised to train the network. With an initial learning rate of $5e^{-4}$, an Adam optimiser and a learning rate scheduler that reduced the learning rate by a factor of 0.5 after observing similar validation loss values for a total of at least 5 epochs were used during the training of the network. The following training and validation loss curves were obtained after training the network for a total of around 130 epochs.

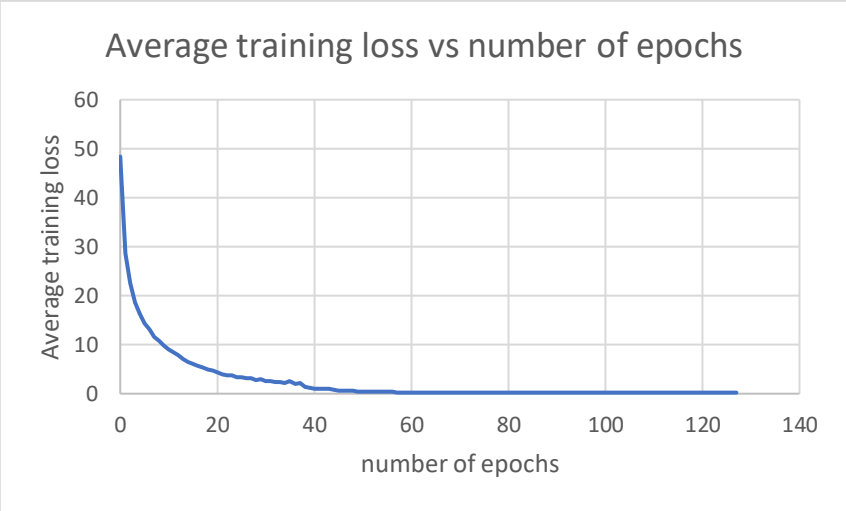

Figure 13: training loss curve observed after training the network with complete dataset of 140,000 samples

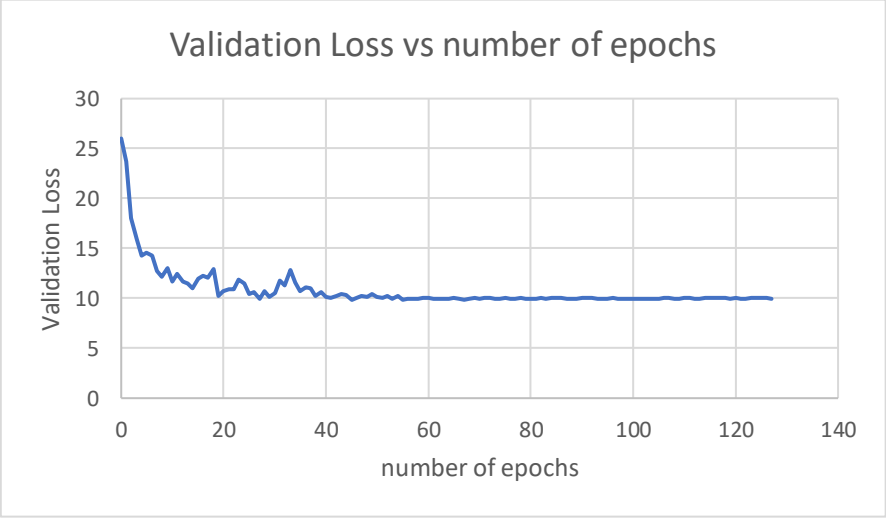

Figure 14: validation loss curve for network after training the network with complete dataset of 140,000 training samples. 10,000 validation samples were used to verify network performance.

It can be noticed from the level at which the validation loss plateaus, the minimum validation loss values obtained around 10 are still relatively high and the network displays significant fluctuations in the validation loss values for different batches of the validation samples.

To justify the fluctuations in the validation loss curve, the network was trained again but the batch size for the validation samples was kept as 1 to observe if particular batches perform worse throughout all the training epochs. After observing the network training process with these parameters, it was noticed that certain samples in the validation dataset constantly incurred a high loss during the training. However, a reason is yet to be found as to why the network incurs a high loss during each epoch for these particular samples.

Chapter 6: Conclusion and Future Work:

The results obtained above after initially training the network suggest a poor performance by the network on the validation dataset due to the fluctuations observed loss values for different batches. Therefore, in comparison to the monocular vision based network, this network currently does not perform well enough.

A clear task to perform after this step would be to conduct more experiments for training the network in order to obtain better hyperparameter settings. During the last experiments conducted on the network training, it was observed that for some particular samples of validation dataset, the network constantly performed very poorly. It would therefore also be beneficial to conduct further experiments to understand the reasons for a poor performance of the network for these samples and to definitively conclude on why the stereo vision based network does not learn the control parameters as effectively as the monocular vision based network.

Further work can also be conducted on experimenting with utilising the stereo vision dataset with a different methodology such as triangulating the object depth from the stereo image pairs and exploiting the depth values to optimise the control of the manipulator.

Bibliography

[1] Zhuang, Zheyu, Jurgen Leitner, Robert Mahony, "Active Perceptual Cognition: Learning Control Lyapunov Function for Real-time Closed Loop Robotic Reaching from Monocular Vision," Canberra, 2018.

[2] A. E. E. T. J. Charles C. Kemp, "Challenges for robot manipulation in human environments [Grand Challenges of Robotics]," *IEEE Robotics and Automation Magazine,* vol. 1, no. 1, pp. 20-29, 2007.

[3] P. I. Corke, "VISUAL CONTROL OF ROBOT - A REVIEW," *World Scientific,* pp. 1-31, 1993.

[4] F. Janabi-Sharif, "Visual Servoing: Theory and Applications," in *Opto-Mechatronic Systems Handbook*, Boca Raton, FL: CRC Press, 2002, pp. 15-1.

[5] Hutchinson, S., Hager, G.D., & Corke, Peter, "A Tutorial on Visual Servo Control," *IEEE Transactions on Robotics and Automation,* vol. 12, no. 6, pp. 651-670, 1996.

[6] H. I. Y. Shirai, "Guiding a robot by visual feedback in assembling tasks," *Pattern Recognition ,* vol. 5, no. 2, pp. 99-108, 1973.

[7] G. Agin, "Computer Vision Systems for Industrial Inspection and Assembly," *Computer: IEEE,* vol. 13, no. 05, pp. 11-20, 1980.

[8] Peter I Corke, Seth A. Hutchinson, "Real-Time Vision, Tracking and Control," in *IEEE, International Conference on Robotics & Automation*, San Fransisco, CA, 2000.

[9] W.J. Wilson ; C.C. Williams Hulls ; G.S. Bell, "Relative end-effector control using Cartesian position based visual servoing," *IEEE Transactions on Robotics and Automation ,* vol. 12, no. 5, pp. 684-696, 1996.

[10] Min-Soo Kim ; Ji-Hoon Ko ; Hee-Jun Kang ; Young-Shick Ro ; Young-Soo Suh, "Image-based manipulator visual servoing using the Kalman Filter," in *2007 International Forum on Strategic Technology*, Ulaanbaatar, Mongolia, 2007.

[11] Luis Montesano, Manuel Lopes, Alexandre Bernardino, Jose Santos-Victor, "Learning Object Affordances: From Sensory–Motor Coordination to Imitation," *IEEE TRANSACTIONS ON ROBOTICS*, vol. 24, no. 1, pp. 15-26, 2008.

[12] ANTONIO MORALES, ERIS CHINELLATO, ANDREW H. FAGG, ANGEL P. DEL POBIL, "USING EXPERIENCE FOR ASSESSING GRASP RELIABILITY," *International Journal of Humanoid Robotics*, vol. 16, p. 20:37, 2004.

[13] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, Carsten Rother, "Learning 6D Object Pose Estimation Using 3D Object Coordinates," *European Conference on Computer Vision, ECCV 2014: Computer Vision – ECCV 2014*, vol. 8690, pp. 536-551, 2014.

[14] Alexander Krull,∗ Eric Brachmann, Frank Michel, Michael Ying Yang, Stefan Gumhold, Carsten Rother, "Learning Analysis-by-Synthesis for 6D Pose Estimation in RGB-D Images," *The IEEE International Conference on Computer Vision (ICCV)*, pp. 954-962, 2015.

[15] Stephen James, Andrew J. Davison, Edward Johns, "Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task," in *1st Conference on Robot Learning (CoRL 2017)*, Mountain View, USA, 2017.

[16] J. L. M. M. P. C. Fangyi Zhang, "Sim-to-real Transfer of Visuo-motor Policies for Reaching in Clutter: Domain Randomization and Adaptation with Modular Networks," 18 september 2017. [Online]. Available: https://128.84.21.199/abs/1709.05746v1. [Accessed 30 april 2019].

[17] Pablo Ramon Soria,* Begoña C. Arrue, and Anibal Ollero, "Detection, Location and Grasping Objects Using a Stereo Sensor on UAV in Outdoor Environments," *Sensors*, vol. 17, p. 103, 2017.

[18] P. F. F. L. (. University), "Convolutional Neural Network," 2018. [Online]. Available: http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/. [Accessed 28 september 2018].

[19] A. M. (. o. B. Saleh Albelwi, "A Framework for Designing the Architectures of Deep Convolutional Neural Networks," Entropy, 2017.

[20] P. F. F. Lee, “CS231n Convolutional Neural Networks for Visual Recognition,” 2018. [Online]. Available: http://cs231n.github.io/. [Accessed August-September 2018].

[21] PyTorch, "Training a Classifier," 2017. [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. [Accessed october 2018].

[22] S. (. Simrock, "Control theory," CAS - CERN Accelerator School: Course on Digital Signal Processing, Sigtuna, Sweden, 2007.

[23] Mohammed Dahleh, Munther A. Dahleh, George Verghese, "Lectures on Dynamic Systems and Control," Massachuasetts Institute of Technology, Masachusettes, 2011.

[24] S.-A. W. H. A. S. N. Parhma M. Kebria, "Kinematic and Dynamic Modelling of UR5 Manipulator," in *2016 IEEE International Conference on Systems, Man, and Cybernetics • SMC 2016* , Budapest, Hungary, 2016.

[25] Calin Andrei Belta, R. Vijay Kumar, "Euclidean metrics for motion generation onSE(3)," University of Pennsylvania Scholarly Commons, 2002.

[26] Weisstein, Eric W., "Frobenius Norm.," MathWorld--A Wolfram Web Resource, [Online]. Available: http://mathworld.wolfram.com/FrobeniusNorm.html .

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. pp. 770-778, 2016.

[28] jia deng, w dong, r socher, jial, li, fei fei l, "ImageNet: A Large-Scale Hierarchical Image Database," *CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition* , pp. pp. 248-255, 2009.

[29] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, Adam Bry, "End-To-End Learning of Geometry and Context for

Deep Stereo Regression," *The IEEE International Conference on Computer Vision (ICCV)*, pp. pp.66-75, 2017.

[30] R. Ruizendaal, "Deep Learning #3: More on CNNs & Handling Overfitting," 12 May 2017. [Online]. Available: https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d.

[31] A. t. P. K. S. R. P. Ulirch Viereck, "Learning a visuomotor controller for real world robotic grasping using simulated depth images," in *Conference on Robotic learning (CoRL)*, Mountain View, 2007.

[32] M. A. W. Z. P. A. Xue Bin peng, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, 2018.

[33] A. H. Nathan Koenig, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.

[34] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software, vol 3, no. 3.2. ,2009*, Kobe, Japan, 2009.

[35] "Chapter 3: FORWARD KINEMATICS: THE DENAVIT-HARTENBERG CONVENTION," [Online]. Available: https://users.cs.duke.edu/~brd/Teaching/Bio/asmb/current/Papers/chap3-forward-kinematics.pdf.

[36] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima," 15 september 2016. [Online]. Available: arXiv:1609.04836v2 [cs.LG] .

[37] Dominic Masters, Carlo Luschi, "Revisiting small batch training for deep neural networks," 28 april 2018. [Online]. Available: arXiv:1804.07612v1 [cs.LG].

[38] Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization," 22 December 2014. [Online]. Available: arXiv:1412.6980v9.

[39] S. M. Lavalle, "Planning Algorithms," 2006. [Online]. Available: http://msl.cs.uiuc.edu/planning/node147.html. [Accessed 2019].

[40] P. Villers, "PRESENT INDUSTRIAL USE OF VISION," in *12th International Symposium on Industrial Robots*, Paris, 1982.