Advancing Visual Servoing Controller for Robotic Manipulation: Dynamic Object Grasping and Real-World Implementation

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering
by Research

by

Gunjan Gupta
2019111035
gunjan.gupta@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
June, 2024

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled **"Advancing Visual Servoing Controller for Robotic Manipulation: Dynamic Object Grasping and Real-World Implementation"** by Gunjan Gupta, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Advisor: Dr. K. Madhava Krishna

To Mummy, Papa and Bhai

Acknowledgments

I would like to express my sincere gratitude towards my advisor, Dr. Madhav Krishna, for his unwavering support and guidance throughout my research journey. Working under his mentorship has been an invaluable experience, I have learned both personally and professionally from him. I also extend my gratitude to Dr. Nagamanikandan Govindan for his invaluable guidance and support in working with real manipulators. I would also like to thank all the RRC professors.

I want to thank M. Nomaan Qureshi, Vedansh Mittal, Pranjali Pathre, P Haasith Venkata Sai, Dr. Samarth Brahmbhatt, Kuustab Pal, Dr. Harit Pandya and Prof. Arun Singh for their outstanding contributions as co-workers. Their expertise, guidance, and availability have been invaluable to me, and I am truly grateful for the opportunity to work alongside such talented individuals. I express my heartfelt appreciation to Shivaan Sehgal for his constant support, motivation, and guidance throughout my journey. His presence has been instrumental in my success, and I am deeply grateful for his constant encouragement and assistance. Additionally, I am grateful to my friends - Sahithi Reddy, Pratishtha Abrol, Vedant Mundheda, Mehul Goyal, Abhijit Srivastava, Aman Kumar Singh, Jay Sharma, Tanmay Garg, KV Aditya Srivastva, Vikrant Dewangan, and Mukund for believing in me.

I am truly indebted to them to my parents for their countless efforts and all the support they have provided me throughout the life. Their guidance and encouragement have been a constant source of strength, shaping me into the person I am today. I also extend my heartfelt thanks to my beloved brother for always being there for me. Their love and dedication have played an integral role in my achievements. Finally, I would like to express my gratitude to all those who have contributed to my growth and development, imparting valuable lessons along the way.

Abstract

Visual servoing been gaining popularity in various real-world vision-centric robotic applications, offering enhanced end-effector control through visual feedback. In the realm of autonomous robotic grasping, where environments are often unseen and unstructured, visual servoing has demonstrated its ability to provide valuable guidance. However, traditional servoing-aided grasping methods encounter challenges when faced with dynamic environments, particularly those involving moving objects.

In the first part of the thesis (**Chapter 3**), we introduce *DynGraspVS*, a novel Visual Servoing-aided Grasping approach that models the motion of moving objects in its interaction matrix. Leveraging a single-step rollout strategy, our approach achieves a remarkable increase in success rate, while converging faster and achieving a smoother trajectory, while maintaining precise alignments in six degrees of freedom (6 DoF). By integrating the velocity information into the interaction matrix, our method is able to successfully complete the challenging task of robotic grasping in the case of dynamic objects, while outperforming existing deep Model Predictive Control (MPC) based methods in the PyBullet simulation environment. We test it with a range of objects in the YCB dataset with varying range of shapes, sizes, and material properties. We show the effectiveness of our approach by reporting against various evaluation metrics such as photometric error, success rate, time taken, and trajectory length.

In addition to introducing DynGraspVS, this thesis in the second half (**Chapter 4**) explores the integration and implementation of Image-Based Visual Servoing (IBVS) mechanisms on the XARM7 robotic platform. Through successful integration, our work demonstrates the feasibility and practical applicability of IBVS in real-world robotic systems. Furthermore, a comprehensive analysis of the Recurrent Task-Visual Servoing (RTVS) framework's performance in diverse real-world scenarios sheds light on its robustness and versatility. Additionally, the introduction of Imagine2Servo, a conditional diffusion model for generating target images, enhances the capabilities of IBVS for complex tasks. Through a combination of experimental validation and rigorous testing, this thesis provides valuable insights into the effectiveness and potential applications of IBVS in real-world robotic systems, setting the stage for future advancements in visual servoing technology.

Contents

Chapter Page

1 Introduction . . . 1
1.1 Contributions . . . 3
1.2 Thesis Layout . . . 3

2 Background . . . 5
2.1 Visual Servoing . . . 5
2.2 LSTM . . . 7
2.2.1 Architecture . . . 7
2.3 Related Work . . . 9
2.3.1 Visual Servoing . . . 9
2.3.2 Deep Visual Servoing . . . 9
2.3.3 Reinforcement Learning based Servoing . . . 9
2.3.4 Model Based Visual Control . . . 10
2.3.5 Dynamic Environment Grasping with Visual Servoing . . . 10
2.3.6 Manipulator . . . 10

3 DynGraspVS: Servoing Aided Grasping for Dynamic Environments . . . 12
3.1 Introduction . . . 12
3.2 Methodology . . . 12
3.2.1 Problem Formulation . . . 14
3.2.2 Deep Network . . . 14
3.2.2.1 Feature Extractor . . . 15
3.2.2.2 Temporal Information and Velocity Prediction . . . 15
3.2.3 Goal Image Generation . . . 15
3.2.3.1 Inference Pipeline . . . 16
3.2.3.2 Image Generation . . . 17
3.2.4 MPC Objective . . . 17
3.2.4.1 Interaction Matrix . . . 18
3.2.4.2 Loss Function . . . 20
3.2.5 Perception System . . . 22
3.2.6 Dynamic Controller . . . 22
3.3 Experiments and Results . . . 23
3.3.1 Simulation Setup . . . 23
3.3.2 Baselines . . . 23
3.3.3 Evaluation Metrics . . . 24

3.3.3.1 Time Taken . . . 24
3.3.3.2 Success Rate . . . 24
3.3.3.3 Trajectory Length . . . 24
3.3.3.4 IoU . . . 24
3.3.3.5 Photometric Error . . . 25
3.3.4 Qualitative Results . . . 26
3.3.5 Quantitative Results . . . 26
3.3.5.1 Object-wise Results . . . 32

4 Real-World Applications . . . 34
4.1 Introduction . . . 34
4.2 Real World Setup . . . 34
4.2.1 xARM 7 . . . 36
4.2.2 Software . . . 37
4.3 Methodology . . . 39
4.3.1 Foresight Model . . . 39
4.3.2 RTVS Controller . . . 41
4.3.3 Inverse Kinematics . . . 41
4.4 Experiments and Results . . . 44
4.4.1 Dataset . . . 44
4.4.2 Evaluating RTVS Efficacy in Real World . . . 45
4.4.3 Results . . . 50
4.5 Challenges . . . 52

5 Conclusions . . . 54

Bibliography . . . 56

List of Figures

Figure Page

1.1 We introduce a solution to the dynamic visual servoing problem, where an object is in motion and needs to be reached. Unlike other approaches, our method remarkably improves trajectory lengths while maintaining precise alignments in 6DoF. With our real-time control generation, the robotic arm successfully reaches its target, outpacing other deep MPC-based visual servoing techniques that struggle to catch up. 2

2.1 **The Fundamental Image-Based Visual Servoing Pipeline:** The process entails the robot's movement guided by control commands, generating a new image. Subsequently, 2D features are extracted from both the current and target image compared to minimize error. Finally, velocity commands are derived from the control generator. 6

2.2 **An LSTM Cell:**Illustrating its critical components and their interactions. The forget gate is highlighted in red, indicating its role in determining which information is retained or discarded from the cell state. The input gate, marked in purple, decides which new information is added to the cell state. The cell state update, depicted in green, shows the combination of retained information and new information added. Lastly, the output gate is highlighted in blue, demonstrating how the cell's output is generated based on the updated cell state. This structure facilitates the LSTM's ability to effectively learn temporal dependencies in sequential data by managing information flow across the network 8

3.1 Here, we present our dynamic visual servoing based grasping architecture, optimising the controls in 6 DoF. We generate the Goal Image, I^* using the Contact Graspnet, which provides the most confident pose to grasp. Subsequently, a Perception System predicts the desired flow, $\mathscr{P}$ between current image, I_t and goal image, I^*. To calculating the objects's velocity relative to the ground, $\vec{v^o}$, we utilise sequence of 5 images - 1 of current timestep and 4 past frames and extract their Depth Images using a Perception System. These RGBD image sequences are then input into a Deep Network, which computes the velocity of object $\vec{v^o}$. Interaction matrix is used along with calculated $\vec{v^o}$ to estimate the Predicted Flow, $\hat{\mathscr{P}}$. The control predictor aims to optimise the perception flow loss, $\mathscr{L}_{flow}$ between the predicted flows and desired flow, $\mathscr{P}$ and generate optimal controls commands, ($\vec{v_t}$, $\vec{w_t}$) to facilitate the grasping process. 13

3.2 **The Deep Network:** Detailed architecture of the proposed deep learning model for 3D object velocity prediction. The model comprises four main components: (1) The Feature Extractor, utilizing a pre-trained and modified ResNet-18 CNN for robust image feature extraction, which captures essential spatial information from a sequence of images through convolutional layers, residual blocks, and global average pooling. (2) The Temporal Features module, employing an LSTM network to process the sequence of 512-dimensional feature vectors extracted from the ResNet-18, enabling the model to understand and encode the temporal dynamics and dependencies across the frames. (3) A fully connected (FC) layer, which translates the high-dimensional LSTM output into a concise 3D vector, encapsulating the object's predicted velocity in the x, y, and z dimensions. (4) The Output, which presents the predicted 3D velocity vector of the object across the sequence. 14

3.3 **The Basic Camera Model:** This figure illustrates the fundamental camera model, depicting the relationship between the camera's position in world coordinates ($\vec{X^c} = [x^c, y^c, z^c]$), the object's position in world coordinates ($\vec{X^o}$) is [X, Y, Z]), and the object's position relative to the camera at point P ($\vec{X_c^o} = [x, y, z]$) in camera frames which corresponds to pixel coordinates (u, v) in the image frame. The rotation matrix, R, transforms coordinates from the world frame to the camera frame, while t = $\vec{X^c}$ represents the translation of the camera within the world frame. 19

3.4 **Qualitative Results on the "Tennis ball" object of YCB dataset**: We show the RGB images overlapped with the associated Flow images for intermittent poses captured during the grasping of the Tennis ball object. Note that the flow uses the color coding mentioned in the supplementary section of FlowNet2.0 [1]. 25

3.5 **Evolution of metrics with time for Banana and Chips Can**: Comparison of Intersection over Union (IoU) and Photometric Error over time for YCB objects Banana and Chips Can at a given velocity. 27

3.6 **Evolution of metrics with time for Mustard Bottle and Tennis Ball**: Comparison of Intersection over Union (IoU) and Photometric Error over time for YCB objects Mustard Bottle and Tennis Ball at a given velocity. 28

3.7 **Velocity Comparison**: This graph shows how velocity varies with number of iterations. 29

3.8 **Trajectory Comparison of object in linear motion**: We plot the trajectory of RTVS, DeepMPCVS and DynGraspVS. 30

3.9 **Trajectory Comparison of object in circular motion**: We plot the trajectory of RTVS, DeepMPCVS and DynGraspVS. 31

3.10 Given is the performance of our baselines on different objects in the YCB dataset. We report (a) IoU, (b) Success Rate (%), (c) Time Taken (s) and (d) Trajectory Length (m). 32

4.1 Complete Setup for working of xARM7 35

4.2 Serial Model of xARM 37

4.3 **Pipeline for Task Execution**: First, the foresight model, a diffusion-based image-to-image translation model conditioned on the current monocular eye-in-hand camera input and additional scene observations, generates the target image. Subsequently, the optical flow between the current and target image is computed to determine the motion required to reach the target. Similarly, the optical flow between the target and the previous image is utilized to predict depth using flow to depth. Once the 6 degrees of freedom (6DOF) control is predicted by the Recurrent Task-Visual Servoing (RTVS) controller, inverse kinematics (IK) is employed to calculate the joint angles necessary to move the manipulator to the predicted end effector position. 40
4.4 **Real-world setup**: We employed a UFACTORY xARM7 robotic manipulator equipped with an eye-in-hand Intel RealSense D455 depth camera. 44
4.5 **Converge Results**: This figures shows that the RTVS is able to converge in the basic setup where there is object on the simple floor. 45
4.6 **Converge Results**: The RTVS results on a complicated environment. 46
4.7 **Comparison of various backgrounds**: The RTVS results variations of backgrounds. . 47
4.8 **Flow Comparison**: This figure shows the comparison between various optical flow methods. 48
4.9 Trajectory demonstrating convergence utilizing flow depth and RAFT. 49
4.10 Qualitative Analysis of Task 1. 51
4.11 Qualitative Analysis of Task 2. 52

List of Tables

Table Page

3.1 **Benchmark Comparison on YCB dataset**: The given table denotes results aggregated over all objects in YCB dataset. Our method outperforms the baselines in all metrics. . 26

4.1 xARM7 Joint Range 36
4.2 Specifications of each link 38
4.3 Standard D-H Parameters of xARM7 43

Chapter 1

Introduction

The intersection of robotics, computer vision, and control systems has given rise to groundbreaking advancements across domains. Visual servoing offers a compelling alternative to traditional methods for predicting robot controls, including reinforcement learning (RL). Unlike kinematic or dynamic models that rely solely on the geometric or dynamic properties of the robot and its environment, visual servoing harnesses the power of vision to guide robot actions [2, 3]. By integrating real-time visual feedback from cameras into the control loop, visual servoing enables robots to perceive and respond to environmental changes with unparalleled flexibility and adaptability [4]. In contrast to RL, which learns control policies through reward and punishment interactions with the environment, visual servoing offers a more structured and deterministic approach to the robot control. While RL can achieve impressive results in tasks with well-defined objectives and ample exploration time, it often struggles in real-time applications where precise control is required and safety is paramount [5]. Visual servoing, on the other hand, leverages explicit visual cues to guide robot actions, enabling precise and responsive control in dynamic environments without extensive training or exploration. By directly incorporating visual feedback into the control loop, visual servoing bypasses the need for complex reward functions and exploration strategies, offering a more straightforward and interpretable approach to robot control. This makes visual servoing particularly well-suited for tasks requiring real-time responsiveness and robustness, such as robotic manipulation in cluttered or dynamic environments.

Motivation: In the context of grasping moving objects, the dynamic and unpredictable nature of the environment poses significant challenges for traditional robotic manipulation techniques. Conventional methods based on pre-defined models need help to adapt to the variability and uncertainty inherent in dynamic scenarios, often resulting in suboptimal performance or failure to grasp moving objects reliably. In contrast, Image-Based Visual Servoing (IBVS) offers a promising solution that directly utilizes visual information to guide grasping actions. By dynamically adjusting robot motions based on real-time visual feedback, IBVS enables robots to respond promptly and accurately to changes in the object's position and orientation, improving the chances of successful grasping in dynamic environments.

In this thesis, we introduce a novel algorithm designed to adapt the visual servoing-aided grasping methods to the shortcomings of dynamic environments. Our proposed approach considers the motion of

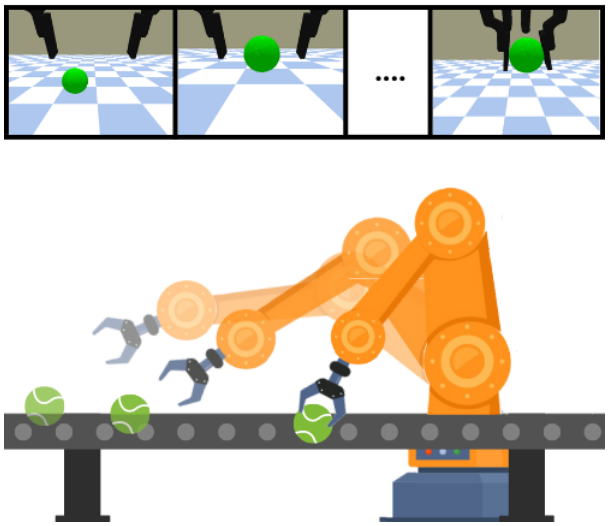


Figure 1.1: We introduce a solution to the dynamic visual servoing problem, where an object is in motion and needs to be reached. Unlike other approaches, our method remarkably improves trajectory lengths while maintaining precise alignments in 6DoF. With our real-time control generation, the robotic arm successfully reaches its target, outpacing other deep MPC-based visual servoing techniques that struggle to catch up.

moving objects and generates optimized control strategies to control the robotic arm. Unlike traditional methods that often struggle with tracking objects in motion [6], our algorithm harnesses predictive models, adaptive control techniques, and real-time visual feedback. Our approach is able to converge and perform the execution of actions needed to reach and manipulate dynamic objects. Leveraging Contact GraspNet [7], we generate desired images from the most confident poses identified through the Contact GraspNet module. Through the Cross-Entropy Method (CEM) [8], we sample controls, resulting in a remarkable success rate exceeding 90% in grasping tasks.

We look into implementing Image-Based Visual Servoing (IBVS) in real-world scenarios to evaluate its adaptability and effectiveness. Our focus lies in harnessing the capabilities of the XARM7 robotic manipulator to integrate IBVS control mechanisms and gauge their performance in practical environments. However, a significant challenge emerged during the implementation of RTVS proposed in [9]: the need for a model capable of generating target images autonomously. In many scenarios, manually capturing target images proves impractical or infeasible, necessitating an automated solution to generate sub-goal images for long-range servoing tasks.

We adopt a diffusion model proposed in [10] to autonomously predict the sub-goal images to address this challenge. With this model, we generated target images essential for predicting the manipulator's control. These target images enabled us to successfully execute tasks like sorting and stacking using xARM7 manipulator, showcasing the effectiveness of our approach in real-world settings.

1.1 Contributions

Our contributions are highlighted through the following key innovations:

- We introduce a novel real-time visual servoing based control algorithm tailored to dynamic environments. This new approach is able to handle the challenges of grasping moving objects by accurately predict controls using 6D pose.
- By introducing the velocity terms in the interaction matrix, our approach achieves shorter trajectories and heightened reachability, outperforming previous approaches. Adding velocity terms in the interaction matrix enhances performance, facilitating accurate interaction with moving objects.
- We conducted extensive experiments involving objects of varying sizes, shapes, and velocities, encompassing linear motions. This inclusive approach mirrors real-world scenarios and accentuates the versatility of our proposed framework.
- Our work includes the successful integration and implementation of IBVS control mechanism [9] on the XARM7 platform, demonstrating the feasibility and practical applicability of IBVS in real-world robotic systems.

1.2 Thesis Layout

C1 In this introductory chapter, we provide an overview of the scope of the work presented in this thesis within the context of advancements in visual servoing for dynamic environment. We identify the key challenges we aim to address and outline the motivation behind the methods we will develop in subsequent chapters.

C2 The second chapter serves as a comprehensive background exploration, focusing on the vast field of visual servoing and manipulators. It is dedicated to reviewing existing literature, where we discuss significant research, methodologies, and the advancements contributed by fellow researchers within these domains.

C3 This chapter focuses on the detailed implementation of DynGraspVS, including its components, methodologies, and experimental setups. We present the experiments conducted and analyze the results obtained from applying DynGraspVS in various scenarios.

C4 In this chapter, we explore the intricacies of implementing RTVS on the XARM 7 robotic manipulator. We discuss the process of integrating RTVS into the hardware platform, highlighting key details of the implementation and addressing any challenges encountered along the way.

C5 The final chapter provides the conclusion to the thesis, summarizing the key findings, contributions, and implications of the research presented. We discuss the significance of our work, its limitations, and potential avenues for future research and development in the field.

Chapter 2

Background

2.1 Visual Servoing

Visual servoing is a technique used in robotics to control the motion of a robotic system using visual feedback obtained from sensors, typically RGB cameras. The primary goal of Visual Servoing is to manipulate the robot's movements based on real-time visual information, allowing it to perform tasks such as object tracking, manipulation, and navigation as shown in 2.1

Within the broader framework of Visual Servoing, two main categories are often distinguished based on the method used to guide the robot's motion:

Image Based Visual Servoing relies on extracting 2D features from RGB images, such as keypoints or edges, to formulate control laws. These control laws are designed to minimize the error between the desired and actual positions of these features in the image frame. By continuously adjusting the robot's motion based on changes in the visual feedback, IBVS enables accurate and responsive control.

Position Based Visual Servoing incorporates both the position of objects within the camera frame and features extracted from RGB images to predict the 3D position of objects in the robot's workspace. This prediction is then used to optimize the robot's control, allowing for precise manipulation and navigation tasks.

The primary objective of Visual Servoing is to minimize the error, denoted as $e(t)$, at time t, between the current features $\mathscr{F}(t)$ and the desired features $\mathscr{F}^*$, which remain constant throughout the process. This error is calculated as 2.1:

$$e(t) = \mathscr{F}(t) - \mathscr{F}^* \tag{2.1}$$

The selection of $\mathscr{F}(t)$ is crucial and requires careful consideration to identify the pertinent features. After selecting the appropriate features, the control law predicts the 6-DOF velocity V_t. The relationship between the features $\mathscr{F}$ and V_t is expressed in 2.2:

$$\dot{\mathscr{F}} = L_f V_t \tag{2.2}$$

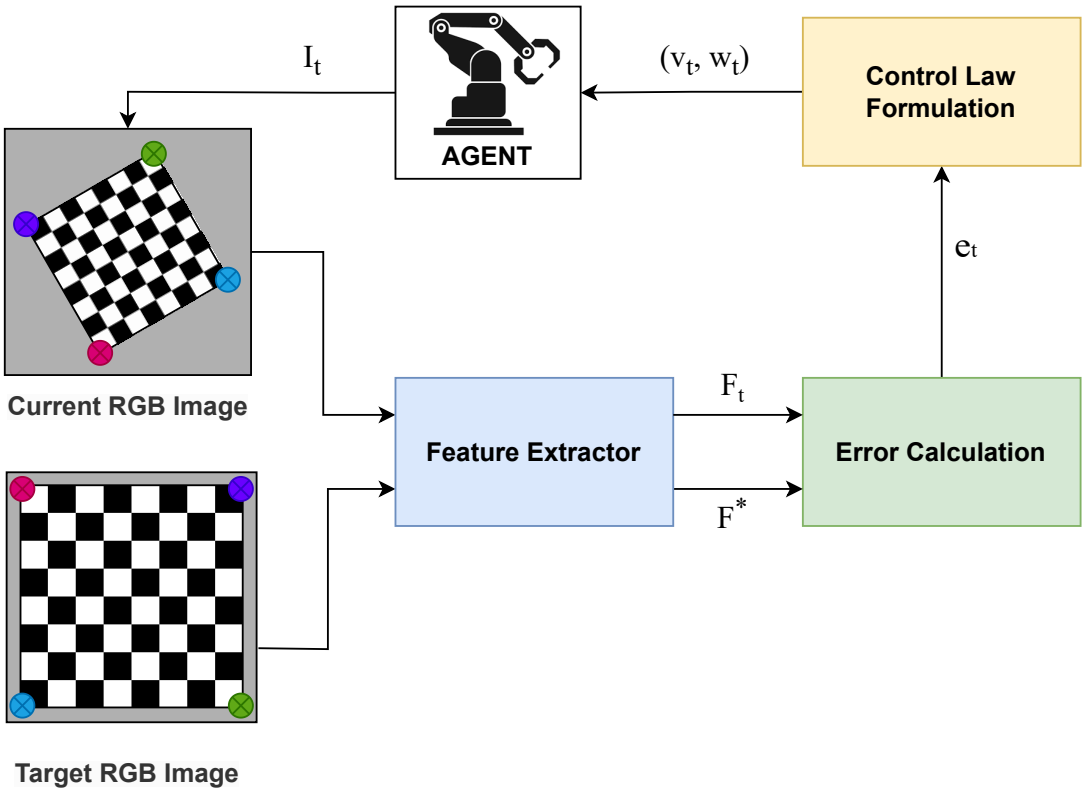


Figure 2.1: **The Fundamental Image-Based Visual Servoing Pipeline:** The process entails the robot's movement guided by control commands, generating a new image. Subsequently, 2D features are extracted from both the current and target image compared to minimize error. Finally, velocity commands are derived from the control generator.

Here, L_f represents the interaction matrix, depicting the correlation between the motion of features and the camera's motion. This relationship forms the basis for deriving effective control strategies in Visual Servoing.

2.2 LSTM

Long Short-Term Memory networks (LSTMs) are a special kind of Recurrent Neural Network (RNN) capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber in 1997 [11] to address the vanishing gradient problem encountered by standard RNNs during backpropagation. LSTMs are designed to maintain a memory over long sequences, making them particularly effective for tasks involving sequential data such as time series prediction, natural language processing, and, as in our case, tracking object movement through a sequence of images.

2.2.1 Architecture

An LSTM unit comprises four main components: the cell state, the input gate, the output gate, and the forget gate. These components work together, allowing the unit to remember or forget information selectively.

Cell State: The cell state, C_t, represents the "memory" of the LSTM unit, carrying relevant information through the sequence.

Forget Gate: The forget gate, f_t defined in 2.3, decides which information should be discarded from the cell state. It uses the previous output h_{t-1} and current input x_t to generate a value between 0 and 1 for each number in the cell state C_{t-1}. The operation is defined as:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.3}$$

Input Gate: The input gate, i_t 2.4, determines which new information will be stored in the cell state. Meanwhile, a tanh layer proposes a new candidate values vector, $\tilde{C}_t$, that could be added to the state using 2.5:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.4}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.5}$$

Output Gate: The output gate, o_t, decides the next hidden state h_t, which contains information about previous inputs. The hidden state is used for predictions and passed to the next LSTM unit using 2.6 and 2.7:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.6}$$

$$h_t = o_t * \tanh(C_t) \tag{2.7}$$

Updating Cell State: The cell state is updated to the new cell state C_t by removing information deemed unnecessary by the forget gate and adding new candidate values scaled by the input gate's output, 2.8:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.8}$$

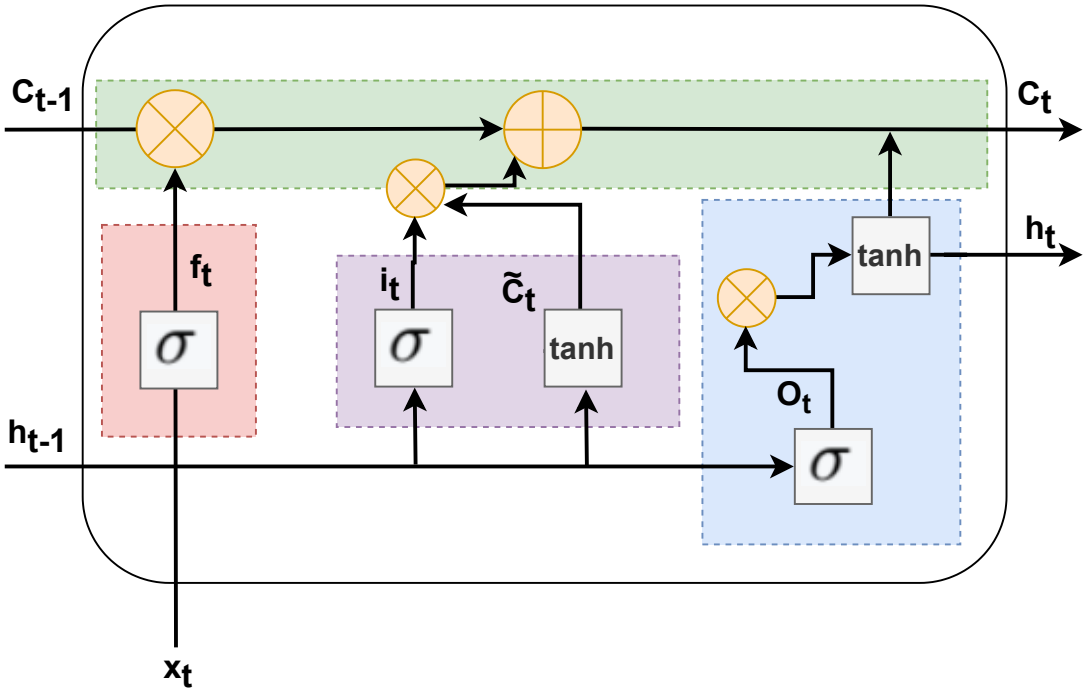


Figure 2.2: **An LSTM Cell:**Illustrating its critical components and their interactions. The forget gate is highlighted in red, indicating its role in determining which information is retained or discarded from the cell state. The input gate, marked in purple, decides which new information is added to the cell state. The cell state update, depicted in green, shows the combination of retained information and new information added. Lastly, the output gate is highlighted in blue, demonstrating how the cell's output is generated based on the updated cell state. This structure facilitates the LSTM's ability to effectively learn temporal dependencies in sequential data by managing information flow across the network

2.3 Related Work

2.3.1 Visual Servoing

Visual Servoing is a well-known approach in robotics, aiming to achieve precise target poses through the utilization of image data from a camera sensor. It was first introduced in 1979 by [12]. Visual Servoing is mainly classified into two control laws, IBVS [13] and PBVS [14]. The Image-based Visual Servoing (IBVS) paradigm centers on explicit feature extraction to minimize errors in image space. Diverse strategies have been employed for feature extraction in IBVS. Keypoint-based methods, like SIFT [3], [15], are notable for their robustness in handling changes in scale and orientation. Edge-based approaches, as evidenced by [16, 17], utilize edge detectors to capture significant contours within the scene, thereby assisting in pose estimation. [14] takes advantage of indirect visual servoing and tries to map the feature space to joint velocities using Jacobian. In contrast, the methods that use direct visual servoing [18] aim to directly minimize the photometric error [19] in the images instead of extracting the features from the images. In these approaches, one might avoid the issue of incorrect matching of features. However, these methods need to improve when confronted with substantial camera transformations.

2.3.2 Deep Visual Servoing

In response to the limitations of classical visual servoing, deep learning-based methods have emerged. Recent approaches like [20, 21] tend to learn the relation between the current and the target image and predict the camera's relative pose, leading to smaller steps. Convolutional Neural Networks (CNN)-based techniques [22], such as those enhancing object recognition [23] and motion estimation [24], have been strengthening the convergence rate, but these supervised approaches lacking the ability to generalize to an unknown environment. The [25] generalizes to unseen environments by combining an unsupervised model with visual servoing to predict the 6-DOF controls. Yet, these approaches primarily focus on predicting single-step future controls, making it subpar as it can get stuck in local minima due to greedy algorithms.

2.3.3 Reinforcement Learning based Servoing

In recent years, there have been a surge in reinforcement learning-based navigation methods aimed at achieving target-reaching tasks [26, 27]. These methods typically segment the low-level policy into subtasks to facilitate efficient decision-making. However, a standard limitation is their prediction of optimal controls, which usually do not encompass all six degrees of freedom (6DoF) over longer horizons. This approach often results in inefficiencies due to the extended time frame required for planning. Additionally, deep reinforcement learning techniques such as [28, 29] aim to learn both the controller and system dynamics simultaneously. While promising, this approach faces challenges adapting to unseen environments and handling higher-dimensional continuous actions.

2.3.4 Model Based Visual Control

Model predictive control (MPC) optimizations are successful in the cases of visual feedback due to the usage of accurate feature mapping. One of the earliest works in MPC-based models proposed with visual servoing [30] uses a few key points with the assumption that key points are matched accurately. Due to the scaling issue in classical approaches of MPC, deep learning-based approaches have been introduced [31, 32]. In [33], the MPC model is formulated to optimize an optical flow-based interaction matrix using the Cross-Entropy Method (CEM), offering a novel approach to motion planning. By utilizing a vanilla neural network for predicting velocity from the flow, their method exhibits potential for generalizability across various scenarios. However, the computationally expensive nature of training the vanilla neural network on the fly limits its practicality for real-time applications. In contrast, RTVS [9] presents a more efficient solution by leveraging an LSTM model and differential CEM, enabling real-time operation without sacrificing performance.

2.3.5 Dynamic Environment Grasping with Visual Servoing

The traditional method for grasping needs to identify the target and perform optimal pose estimation [34], but it struggles to generalize effectively in real-world scenarios. [35] is one of the first deep-learning based approaches used for grasping. It achieved 73.9% accuracy. However, the drawback of the approach is the very slow inference speed. The latest approaches do not incorporate visual feedback, making them impractical to adapt to the real-world. There exists previous approaches [36], [37] addressing the challenge of reaching moving objects. The technique proposed by Weiss et al. [37], based on the image Jacobian matrix, holds a prominent place in robot hand-eye coordination. Its advantages lie in addressing the non-linear relationship between image and spatial coordinates. Despite its merits, this technique falls short in certain contexts. Ribeiro et al., [38] proposes a pipeline to grasp moving objects using CNN based visual servo control. It needs a training the CNN to predict the controls.

2.3.6 Manipulator

Manipulators are increasingly employed across various industries for their unparalleled ability to manipulate objects with precision and efficiency. From manufacturing [39] and assembly lines to warehouse logistics [40], agriculture [41], space exploration [42], garbage management [43] and healthcare [44] settings, manipulators play a crucial role in automating tasks and streamlining processes. Their versatility and adaptability make them indispensable tools in modern-day operations. Many studies have been performed to understand the motion dynamics of manipulators, typically categorized into two main types: forward and inverse kinematics [45]. According to Dereli et al., [46], inverse kinematics poses a more challenging task than forward kinematics, primarily due to its involvement with nonlinear equations. These equations are complex to solve, with instances where no feasible solution exists.

Manipulators with 7 degrees of freedom (DOF) hold a distinct advantage over non-redundant manipulators, as they possess the ability to navigate away from singularity configurations with greater ease [47]. However, despite this advantage, the inverse kinematics of 7 DOF manipulators remain complex, as discussed by [48]. In solving inverse kinematics problems, two main methods are commonly employed: analytical methods [49], which involve deriving closed-form solutions, and numerical methods [50], which utilize iterative techniques to approximate solutions. [51] explains that the analytical methods are fast, but they are only applicable to a few special structures. The article [52] provides a comprehensive overview of the Jacobian-based mathematical model for inverse kinematics.

Significant advancements have been made in this field. For instance, [53] proposes a novel numerical method for solving the inverse kinematics problem of 7-DOF redundant manipulators while considering self-collision avoidance. This method combines the Jacobian pseudo-inverse with a penalty function approach to ensure the manipulator avoids collisions with itself during movement. Similarly, [54] addresses singularity avoidance and posture optimization for 7-DOF redundant manipulators by modifying the traditional Jacobian pseudo-inverse method. This modification considers these factors, ensuring the manipulator avoids singular configurations and maintains a preferred posture during movement.

More recently, deep learning techniques have been applied to inverse kinematics. For example, [55] explores a Deep Reinforcement Learning (RL) approach for robot arm control. In this approach, the RL agent learns to solve the inverse kinematics problem and achieve desired end-effector poses through trial and error in a simulated environment. This method considers factors such as joint limits and path smoothness to achieve accurate and efficient manipulation tasks.

In the domain of robotic manipulation, especially in dynamic environments, a notable gap is evident in research focusing on grasping using visual servoing for moving objects using monocular camera. The absence of prior research tackling this specific challenge underscores the novelty and significance of our approach. We extend [9] working to incorporate dynamic environments by introducing velocities in interaction matrix.

Chapter 3

DynGraspVS: Servoing Aided Grasping for Dynamic Environments

3.1 Introduction

This chapter introduces the DynGraspVS model, a novel approach for achieving robust grasping in dynamic environments. The increasing demand for robots to operate in real-world scenarios necessitates controllers that can handle unpredictable object motion. Traditional visual servoing techniques often struggle in such situations, as they primarily rely on the current camera image and don't explicitly account for object dynamics.

To address these challenges, we introduce an Image-Based Visual Servoing (IBVS) model specifically designed for dynamic settings. Our method builds upon and refines the concepts introduced in [9], notably by integrating object velocity into the interaction matrix, a critical advancement that significantly boosts the system's responsiveness to moving objects.

The foundation of this chapter is laid by briefly revisiting the fundamentals of Visual Servoing and the conventional model of IBVS. We then provide an overview of our approach, paving the way for a detailed exploration of each architectural component within subsequent sections of 3.2.

Following the methodology exposition, the chapter progresses to discuss experimental setups and result evaluations. A pivotal aspect of our experimental framework is the use of the PyBullet [56] simulation environment, coupled with a UR5e robotic arm known for its precision and adaptability. This setup is further augmented with objects from the YCB [57] dataset, chosen for their diverse range of attributes, providing a comprehensive testing ground to rigorously assess the efficacy of our dynamic visual servoing model in interacting with moving objects. Through this structured approach, we aim to demonstrate the viability and effectiveness of DynGraspVS in advancing the capabilities of robotic systems in real-world, dynamic scenarios.

3.2 Methodology

We propose an architecture (Fig. 3.1) which takes RGB image, I_t as an input and predicts the velocity, v_t of the robotic arm. The components of the model are described in detail below.

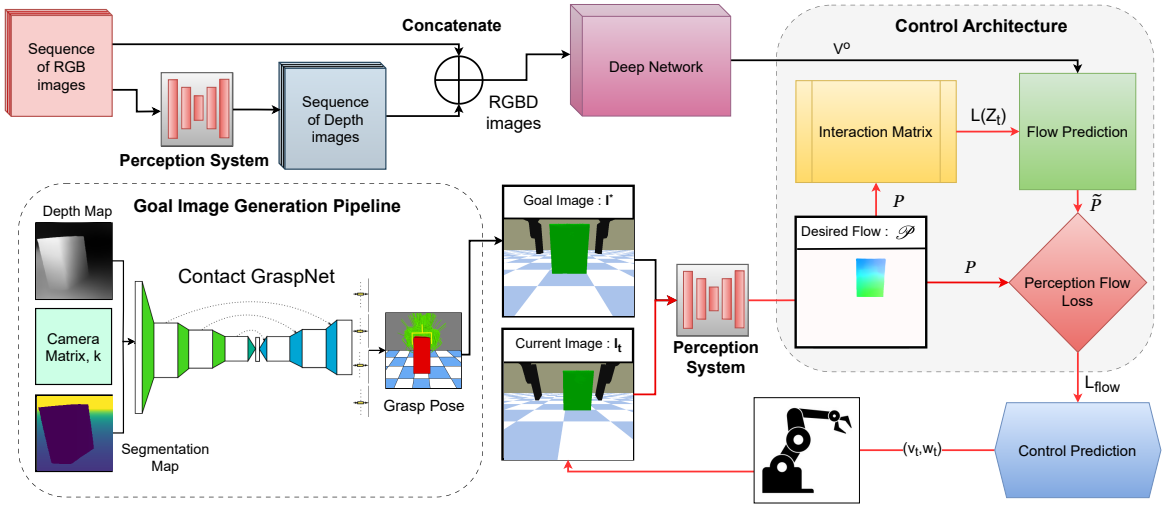


Figure 3.1: Here, we present our dynamic visual servoing based grasping architecture, optimising the controls in 6 DoF. We generate the Goal Image, I^* using the Contact Graspnet, which provides the most confident pose to grasp. Subsequently, a Perception System predicts the desired flow, $\mathscr{P}$ between current image, I_t and goal image, I^*. To calculating the objects's velocity relative to the ground, $\vec{v^o}$, we utilise sequence of 5 images - 1 of current timestep and 4 past frames and extract their Depth Images using a Perception System. These RGBD image sequences are then input into a Deep Network, which computes the velocity of object $\vec{v^o}$. Interaction matrix is used along with calculated $\vec{v^o}$ to estimate the Predicted Flow, $\hat{\mathscr{P}}$. The control predictor aims to optimise the perception flow loss, $\mathscr{L}_{flow}$ between the predicted flows and desired flow, $\mathscr{P}$ and generate optimal controls commands, ($\vec{v_t}$, $\vec{w_t}$) to facilitate the grasping process.

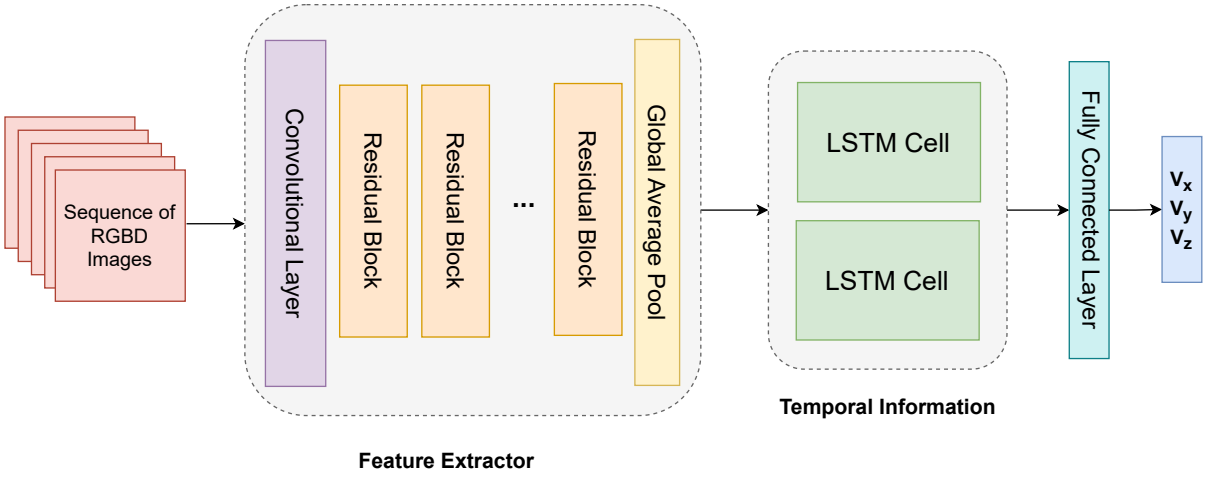


Figure 3.2: **The Deep Network:** Detailed architecture of the proposed deep learning model for 3D object velocity prediction. The model comprises four main components: (1) The Feature Extractor, utilizing a pre-trained and modified ResNet-18 CNN for robust image feature extraction, which captures essential spatial information from a sequence of images through convolutional layers, residual blocks, and global average pooling. (2) The Temporal Features module, employing an LSTM network to process the sequence of 512-dimensional feature vectors extracted from the ResNet-18, enabling the model to understand and encode the temporal dynamics and dependencies across the frames. (3) A fully connected (FC) layer, which translates the high-dimensional LSTM output into a concise 3D vector, encapsulating the object's predicted velocity in the x, y, and z dimensions. (4) The Output, which presents the predicted 3D velocity vector of the object across the sequence.

3.2.1 Problem Formulation

Given a monocular RGB image I_t captured from an in-motion robotic arm at any time instant t, the goal image I^* is generated from ContactGraspNet of the object that is relevant to perform a specific grasping task. The aim is to finally generate a set of optimal control commands $[v_t, \omega_t]$ necessary for the robotic arm to reach the moving object appropriately. The object is moving linearly with a speed of $\vec{v^o}$. The task is performed by minimizing the photo-metric error, $e_t = ||I_t - I^*||$, the basic objection function proposed for Image-based visual servoing (IBVS).

3.2.2 Deep Network

We propose a deep learning architecture, as shown in 3.2, for object tracking that tackles the challenge of predicting the 3D velocity of objects within a sequence of images. The architecture, illustrated

in Figure 1, leverages the complementary strengths of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to achieve this goal.

3.2.2.1 Feature Extractor

The cornerstone of the proposed architecture is a pre-trained modified ResNet-18 model employed for feature extraction. ResNet-18 has established itself as a powerful tool for image classification tasks, demonstrating its ability to learn informative representations from visual data. In this context, the model is harnessed to extract meaningful features that effectively represent the objects present in each frame of the sequence of images.

The initial stage of feature extraction involves a convolutional layer with a 7x7 kernel size, stride of 2, and padding of 3. This layer performs downsampling while capturing low-level features such as edges and corners that are crucial for object recognition. Subsequently, the core of ResNet-18 consisting of residual blocks takes center stage. These residual blocks address the vanishing gradient problem, a common challenge in deep learning, by building upon the outputs of previous layers. This allows the network to learn increasingly complex feature representations that capture more intricate relationships within the data. Finally, Global Average Pooling is applied after the final residual block. This operation transforms the feature maps extracted from each frame into a fixed-size vector of 512 dimensions, essentially summarizing the essential information about the objects present in that frame.

3.2.2.2 Temporal Information and Velocity Prediction

A critical aspect of accurately predicting object velocity lies in understanding the temporal dependencies between frames. This is where the power of Long Short-Term Memory (LSTM) networks comes into play. LSTMs are a type of recurrent neural network specifically designed to handle sequential data. In this architecture, the LSTM module receives the sequence of feature vectors generated by the ResNet-18 encoder. Each vector in the sequence represents the extracted features from a single frame. The LSTM then processes this sequence by analyzing how the features change over time. This allows the network to capture the dynamics of object movement across frames, which is crucial for accurate velocity prediction.

LSTM is connected to a fully connected layer that transforms the output of the LSTM network into a 3D vector. This vector represents the predicted velocity of the object in the world frame for the entire image sequence in x, y and z direction.

3.2.3 Goal Image Generation

This section explains how our framework utilizes Contact GraspNet's [7] output to generate the **goal image** that guides our visual servoing controller.

3.2.3.1 Inference Pipeline

The Contact GraspNet is a sophisticated deep learning model designed to predict the distribution of 6 Degrees of Freedom (6 DOF) grasp poses for robotic manipulation tasks.

- **Input**: The object data, including depth map (or point cloud), camera matrix (K), and optional segmentation map, is formatted as an .npz or .npy file for efficient processing.

- **Pre-processing**: Internally, Contact GraspNet's inference pipeline might incorporate a depth-to-point cloud conversion module, in case depth map as input. This module utilizes the depth map (D) and camera matrix (K) to reconstruct a 3D point cloud (P) of the object:

$$X = \frac{(u - u_0) \cdot Z}{f_x} \tag{3.1}$$

$$Y = \frac{(v - v_0) \cdot Z}{f_y} \tag{3.2}$$

$$Z = D(u, v) \tag{3.3}$$

  where,

  - (u, v) are pixel coordinates in the depth map,
  - (u_0, v_0) is the principal point from the camera matrix (K),
  - f_x and f_y are the focal lengths,
  - X, Y, and Z represent the 3D coordinates of a point in the object,
  - and $D(u, v)$ is the depth value retrieved from the depth map for that pixel location.

  Once the point cloud (P) is reconstructed, it becomes the primary input for Contact GraspNet's internal network architecture. The network likely employs a CNN for feature extraction from the point cloud data, followed by a lightweight module for predicting a distribution of grasp poses.

- **Top Grasp Pose**: The raw output from Contact GraspNet is a distribution of grasp poses, each with a 6DOF configuration (g = [R, t]) and additional information like gripper width, w, openings, go, contact point, c, and a confidence score, CS. Non-Maximum Suppression (NMS) is applied as a post-processing step to remove redundant grasp predictions in close proximity. A threshold (τ) is applied to the grasp confidence scores (CS_i) to filter out low-confidence grasps:

$$G_{filtered} = g_i | CS_i \geq \tau \tag{3.4}$$

  From the remaining grasps, the most confident grasp is chosen based on its score. In case of ties, the contact point might be used as a secondary criterion.

3.2.3.2 Image Generation

Once the final grasp pose (g*) is selected, it defines the desired gripper configuration relative to the object. This includes the rotation matrix (R*) and translation vector (t*) defining the gripper's orientation and position relative to the object.

$$R^* = [\vec{a}, \vec{b} \times \vec{a}, \vec{b}] \tag{3.5}$$

$$t^* = c^* + \frac{w}{2}\vec{a} + d\vec{b} \tag{3.6}$$

Here,

- $\vec{b}$: Represents the approach direction, indicating the direction along which the gripper fingers close on the object.
- $\vec{a}$: Represents the axis of the gripper, defining the orientation of the gripper fingers.
- d: The distance along the approach direction, $\vec{b}$ between the gripper's center point and the contact point on the object.
- c*: Contact Point of the gripper and the object.

Apply the grasp pose (g*) to each vertex (v_i) of the object model to transform it into the desired grasp configuration, v_i' as mentioned in 3.7

$$v_i' = R(v_i) + t^* \tag{3.7}$$

Project each transformed vertex (v_i') onto the image plane, pixels (u_i, v_i) using the camera matrix (K) by using the formula 3.8

$$(u_i, v_i) = K * [v_{ix}', v_{iy}', v_{iz}', 1] \tag{3.8}$$

For each pixel location, maintain a depth buffer by storing the depth value (z-coordinate) of the closest object surface point projected onto that pixel to prevent occlusions. Iterate through all projected vertices, If the current vertex depth ($v_i'z$) is closer than the corresponding value in the depth buffer ($Z(u_i, v_i)$), update the depth buffer: $Z(u_i, v_i) = v_i'z$.

The resulting image is the goal image, I* projected from the robot's camera perspective.

3.2.4 MPC Objective

The MPC objective function serves as a dynamic decision-making tool by continuously evaluating the cost function based on current state information and the desired outcome. The MPC system refines control actions in real-time, enabling optimal performance within the defined constraints. The success of robotic grasping tasks involving moving objects hinges on the ability to adapt to the object's dynamic

behavior in real-time. This section delves into the core of our approach - the Model Predictive Control (MPC) objective function. This function plays a crucial role in guiding the robot's motion towards a successful grasp by continuously minimizing the discrepancy between the observed object motion and the predicted motion based on potential control actions. We build our approach upon the model proposed in RTVS [9], which is futher built upon the MPC objective. We further extend it to the case where both of the robot and the object are in motion and test it for a particular task of grasping.

The equation 3.9 essentially searches for the optimal robot velocity (v_t^*) that minimizes the discrepancy between the perceived flow (actual object motion) and the scaled the pseudo-flow, $\hat{\mathscr{P}}$ which is calculated by eq. 3.10 as given below:

$$v_t^* = \underset{v_t}{\text{ArgMin}}(\left\| \mathscr{P}(I_t, I^*) - h * \hat{\mathscr{P}}(v_t) \right\|) \tag{3.9}$$

Perceived flow($\mathscr{P}(I_t, I^*)$) embodies the actual motion of the object observed by the camera system. It's typically calculated using Image-based Visual Servoing (IBVS) techniques that analyze the current image (I_t) captured by the camera and the desired goal image (I*) whereas the pseudo-flow represents the system's estimation of how the object's motion will appear in the camera's view based on a potential control action (v_t) for the robot. In simpler terms, it's a forecast of the object's movement in the image plane if the robot moves with a specific velocity.

Here, h is the scaling factor, adapting the predicted flow to learn the mean optical flow, rather than spanning a horizon of time. It helps to ensure that the MPC controller does not generate large, unrealistic flows for one step rollout. We need to do a one step rollout because it is very difficult to predict the object flow multi-steps in the future.

$$\hat{\mathscr{P}}(v_t) = L(Z_t) * v_t + \frac{1}{Z_t} * v_o \tag{3.10}$$

The calculation of pseudo-flow relies on current velocity of camera, interaction matrix, depth and velocity of object . The **robot's current velocity,** v_t is typically a 3 DoF vector and the **interaction matrix,** $L(Z_t)$ explains how depth information influences the flow and defines a relationship between features from camera frame to image frame. The multiplication of v_t with $L(Z_t)$ transforms the raw velocity into a weighted velocity. This weighted velocity reflects the expected motion of the object in the image plane, taking into account its distance from the camera forming the foundation of the pseudo flow.

3.2.4.1 Interaction Matrix

This section introduces the generalized formulation of the interaction matrix to handle moving objects, enabling the system to adapt to dynamic environments. the interaction matrix is concerned with the relationship between the motion of objects in the camera's field of view and the resulting motion observed in the camera's image plane. This relationship is influenced by several factors, including the

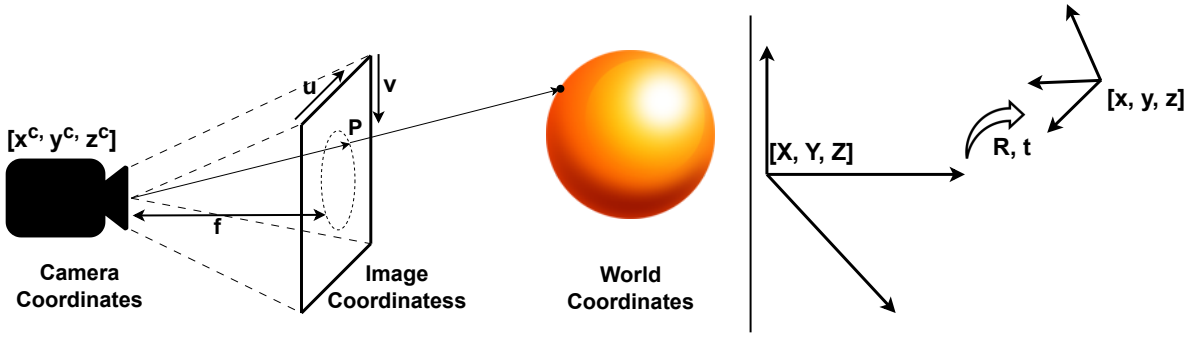


Figure 3.3: **The Basic Camera Model:** This figure illustrates the fundamental camera model, depicting the relationship between the camera's position in world coordinates ($\vec{X^c} = [x^c, y^c, z^c]$), the object's position in world coordinates ($\vec{X^o}$) is [X, Y, Z]), and the object's position relative to the camera at point P ($\vec{X_c^o} = [x, y, z]$) in camera frames which corresponds to pixel coordinates (u, v) in the image frame. The rotation matrix, R, transforms coordinates from the world frame to the camera frame, while t = $\vec{X^c}$ represents the translation of the camera within the world frame.

object's depth in the scene (Z_t) at time t, its velocity relative to the camera ($\vec{v_c^o}$), the velocity of the camera ($\vec{v^c}$) and the velocity of object in ground itself, ($\vec{v^o}$).

The formulation to generalise Interaction matrix for moving objects:

$$\vec{X^o} = R * \vec{X_c^o} + \vec{X^c} \tag{3.11}$$

So, R refers to Rotational matrix corresponding to the transformation from camera to ground frame, $\vec{X^o}$ refers to position of object with respect to ground, $\vec{X_c^o}$ refers to position of object with respect to camera and $\vec{X^c}$ refers to position of camera with respect to ground in 3.11.

Assuming that R is constant for a step and differentiating both sides,

$$\vec{X_c^o} = R^{-1}(\vec{X^o} + \vec{X^c}) \tag{3.12}$$

$$\dot{\vec{X}}_c^o = R^{-1}(\dot{\vec{X}}^o + \dot{\vec{X}}^c) \tag{3.13}$$

$$\dot{\vec{X}}_c^o = R^{-1}(\vec{v^o} + \vec{v^c}) \tag{3.14}$$

Let's say, $\vec{v^o} = [p, q, r]$ and $\vec{v^c} = [v_x, v_y, v_z]$. So now, the 3.14 can be re-written as,

$$\vec{X_c^o} = R^{-1} \begin{bmatrix} p - v_x \\ q - v_y \\ r - v_z \end{bmatrix} = \begin{bmatrix} X_c^o{}_x \\ X_c^o{}_y \\ X_c^o{}_z \end{bmatrix} \tag{3.15}$$

Convert the conventional coordinates (focal plane) to normalised homogeneous image coordinates, (pixels-(u,v)) as shown in 3.20.

$$u = \frac{X_c^o}{Z_c^o} \tag{3.16}$$

$$v = \frac{Y_c^o}{Z_c^o} \tag{3.17}$$

So, differentiating both sides in 3.20,

$$\dot{u} = \frac{\dot{X}_c^o}{Z_c^o} - \frac{X_c^o * \dot{Z}_c^o}{Z_c^{o2}} \tag{3.18}$$

$$\dot{u} = R^{-1}(\frac{1}{Z}(p - v_x) - \frac{X_c^o}{Z^2}(r - v_z)) \tag{3.19}$$

$$\dot{u} = R^{-1}(\frac{1}{Z}(p - v_x) - \frac{u}{Z}(r - v_z)) \tag{3.20}$$

We get a similar equation for v, the $\vec{p} = [\dot{u}, \dot{v}]$. Under the assumption that the camera is aligned with the ground, we achieve eq. 3.24.

$$\dot{p} = \frac{1}{Z}\begin{bmatrix} p - v_x + u(v_z - r) \\ q - v_y + v(v_z - r) \end{bmatrix} \tag{3.21}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} -1 & 0 & x \\ 0 & -1 & y \end{bmatrix}\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} + \frac{1}{Z}\begin{bmatrix} p - ur \\ q - vr \end{bmatrix} \tag{3.22}$$

$$L(Z_t) = \frac{1}{Z_t}\begin{bmatrix} -1 & 0 & u \\ 0 & -1 & v \end{bmatrix} \tag{3.23}$$

Substituting eq. 3.23 in eq. 3.22,

$$\vec{v_c^o} = L(Z) * \vec{v^c} + \frac{1}{Z}\vec{v^o} \tag{3.24}$$

The interaction matrix solely depends on the pixels, u, v and depth at time t, Z_t

3.2.4.2 Loss Function

The loss function formulated is to facilitate the optimal control generation is given by eq. 3.25:

$$\mathscr{L}_{\text{flow}} = \|h(L(Z_t) * v_t + \frac{1}{Z_t} * v_o) - \mathscr{P}(I_t, I^*)\| \tag{3.25}$$

The loss function ensures that the MPC controller generates predicted flows that are close to the target flows, while also avoiding unrealistic flows. The presented formulation offers an advantage over the classical IBVS controller by providing an optimal controls.

Algorithm 1 Visual Servoing based Grasping for Dynamic Environment

Require: Current observation I_t, Convergence constant ϵ

Initialise v_o ▷ *Initialise with random velocity*

Generate I^* ▷ *Goal Image from Contact GraspNet*

while $||I_t - I^*|| \leq \epsilon$ **do** ▷ *Convergence Criteria*

  $I_{t-4} : I_{t-1} :=$ GetPrevObs() ▷ *Get 4 previous RGB images*

  $I_t :=$ GetCurrentObs() ▷ *Capture current RGB image from sensor*

  $\mathcal{F}_t :=$ predict-target-flow (I_g, I_t) ▷ *Predict the flow using the flow network*

  $D_{t-4} : D_t := \mathscr{P}(I_{t-4} : I_t)$ ▷ *Predict sequence of Depth images*

  $\vec{v^o} :=$ DeepNetwork$(D_{t-4} : D_t)$ ▷ *Calculate world frame velocity of object*

  $L_t :=$ CalculateInteractionMatrix(Z_t) ▷ *Calculate the Interaction Matrix*

  $v_o =$ MDCEM(v_0) ▷ *Sample the manipulator velocity*

  $\hat{\mathscr{P}} :=$ CalculatePredictedFLow$(L_t, \vec{v^o}, v_o, \mathscr{P}(I_t, I^*))$ ▷ *Predict the flows*

  $v_t = g(v_{t-1})$ ▷ *Compute Controls*

  $\mathscr{L}_{flow} := \|h(L_t * v_t + 1/Z * v_o) - \mathscr{P}(I_t, I^*)\|$ ▷ *Minimise the flow loss*

  $v_{t+1} := min \mathscr{L}_{flow}$ ▷ *Control Commands*

end while

3.2.5 Perception System

The perception system functions as an advanced global pose generator, creating an array of poses. By strategically selecting two adjacent poses and analyzing their differential, it efficiently extracts information pertaining to optical flow or depth, contingent upon the specified requirements. The system utilizes FlowNet2 [1], a deep learning architecture specifically designed for estimating optical flow between consecutive images. FlowNet2 analyzes the visual content of the current image (I_t) and the desired image (I_t^*) computes the displacement of pixels between these images, effectively capturing the motion patterns. The algorithm works by extracting features from each image and then correlating these features across the two images to estimate the flow vectors. This process results in a dense optical flow field, where each vector in the field indicates the direction and magnitude of motion for a corresponding pixel in the first image.

Building on the capabilities of optical flow estimation, the perception system employs an innovative approach to infer depth information from the computed flow. This is based on the principle that the relative motion between the observer and objects within the scene can be indicative of the objects' distances from the observer. Specifically, objects that appear to move faster (indicating larger flow magnitudes) are generally closer to the camera, while objects that appear to move slower (smaller flow magnitudes) are further away. The perception system utilizes the formula, $Z \simeq 1/||flow||$ to calculate the normalized depth, Z, from the optical flow magnitude. This formula encapsulates the inverse relationship between flow magnitude and depth, providing a mechanism to estimate the depth of objects within the scene from two-dimensional image data.

3.2.6 Dynamic Controller

The high-dimensional nature of our flow-based state representations poses a challenge for conventional Model Predictive Control (MPC) solvers, potentially limiting their efficacy in optimizing equation 3.9. To address this challenge, we generate the modified and differential version of Cross Entropy Method [58], MDCEM. The original MPC objective function involves optimizing a continuous control variable, v_t. Instead, we can define a set of N discrete control actions $(v_1, v_2, ..., v_N)$ representing the robot's possible velocities. This discretization simplifies the problem for MDCEM. Mathematically, we replace the continuous optimization problem with a discrete selection problem:

$$v_t^* = \underset{v_i}{\text{ArgMin}}(\|h(L(Z_t) * v_i + \frac{1}{Z_t} * v_o) - \mathscr{P}(I_t, I^*)\| | i = 1, 2, ..., N) \tag{3.26}$$

In our approach, the MDCEM framework is harnessed to strategically choose elite samples, optimizing control actions over time. We draw from the rich range of elite samples to guide our control decisions toward optimal trajectories achieved by the following:

Initially, MDCEM assigns equal probability to all possible control actions in the discretized space, treating each action as equally likely to succeed. This allows the method to explore the entire range of

possibilities initially. Subsequently, each action is evaluated based on the MPC objective function and identifies a subset of best performing actions. MDCEM uses the information from the elite actions to refine its selection process for the next round. It does this by refitting a Gaussian probability distribution over all control actions. This function assigns higher probabilities to the actions that were part of the elite group in the previous round, effectively favoring those that led to better grasping outcomes in the past. At each step of MPC, the hyper parameters of gaussian function is updated making it adaptable to learn the entire process successfully.

3.3 Experiments and Results

This section provides a detailed overview of the simulation setup, baseline comparisons, evaluation metrics, and a comprehensive discussion of both quantitative and qualitative results derived from a series of experiments designed to assess the performance of our proposed models. Through this structured evaluation, we aim to illustrate the effectiveness, adaptability, and potential of our approaches in various scenarios, offering insights into their practical applicability and areas for future enhancement.

3.3.1 Simulation Setup

We conducted a series of experiments in a simulated environment using the PyBullet [56] physics engine. The PyBullet allows for precise control over robot movements, interactions, and sensory data. The simulation environment was populated with diverse set of objects from the YCB [57] dataset, representing a wide range of shapes, sizes, and material properties. We employed a Universal Robots UR5e robotic arm with a robotiq 2f140 gripper, equipped with a high-resolution camera mounted on its end effector, enabling it to capture real-time visual information with the frequency of 25Hz. We designed a set of experimental scenarios to evaluate the performance of our dynamic visual servoing algorithm. Each scenario involved the robotic arm interacting with a moving object, simulating scenarios that robots might encounter in real-world applications.

We trained the Deep Network on a custom toy-dataset, which consisted of GT images and velocities. We use the pretrained models of FlowNet2.0 and ContactGraspNet on the YCB dataset. The training of Deep Network, inference of the pretrained models and the pybullet simulation interface was done on an Nvidia 1080Ti with 40 CPUs.

3.3.2 Baselines

We use the following baselines:

1. DeepMPCVS-aided Grasping (**DeepMPCVS**): We adopt the DeepMPCVS [33] paper to the problem of grasping. DeepMPCVS employs a vanilla neural network as a baseline for MPC problem solving. Despite its ability to achieve real-time convergence and operate in a receding horizon

fashion, the method is hampered by significant computational overheads, limiting its applicability in real-world scenarios.

2. RTVS-aided Grasping (**RTVS**): We apply the Real-Time Visual Servoing (RTVS) approach to address the grasping problem. RTVS, as outlined in the referenced paper [9], does not consider moving objects due to the absence of velocity terms in its kinetic interaction matrix. This limitation arises from the design of the RTVS framework, where the kinetic interaction matrix is used for the dynamic environments.

3.3.3 Evaluation Metrics

We detail the metrics used to compare our performance against the baselines.

3.3.3.1 Time Taken

We use Time Taken(s) as one of the convergence measure. We gain insights into the computational efficiency and responsiveness of our approach. A lower time taken indicates that our method can quickly adapt to changes in the environment and execute the grasping action in a timely manner. This metric is particularly important in real-world applications where fast and responsive robotic actions are necessary.

3.3.3.2 Success Rate

Success Rate measures the effectiveness of our approach in successfully grasping moving objects. It quantifies the percentage of grasping attempts that result in a successful interaction with the object. A higher success rate indicates that our algorithm can reliably grasp moving objects, demonstrating its robustness and effectiveness in dynamic environments.

3.3.3.3 Trajectory Length

Trajectory Length evaluates the efficiency of our approach by measuring the length of the robotic arm's trajectory during grasping interactions. A shorter trajectory length suggests that our algorithm can plan and execute more direct and efficient paths to reach the desired grasp location. This metric provides insights into the algorithm's ability to navigate complex environments and avoid unnecessary movements, ultimately contributing to improved performance and resource utilization.

3.3.3.4 IoU

The Intersection over Union measures the congruence between the predicted grasp region and the desired grasp region. This metric is required due to the continuously changing background, not necessarily matching with the background of goal image, leading to incorrect photometric error. A higher IoU

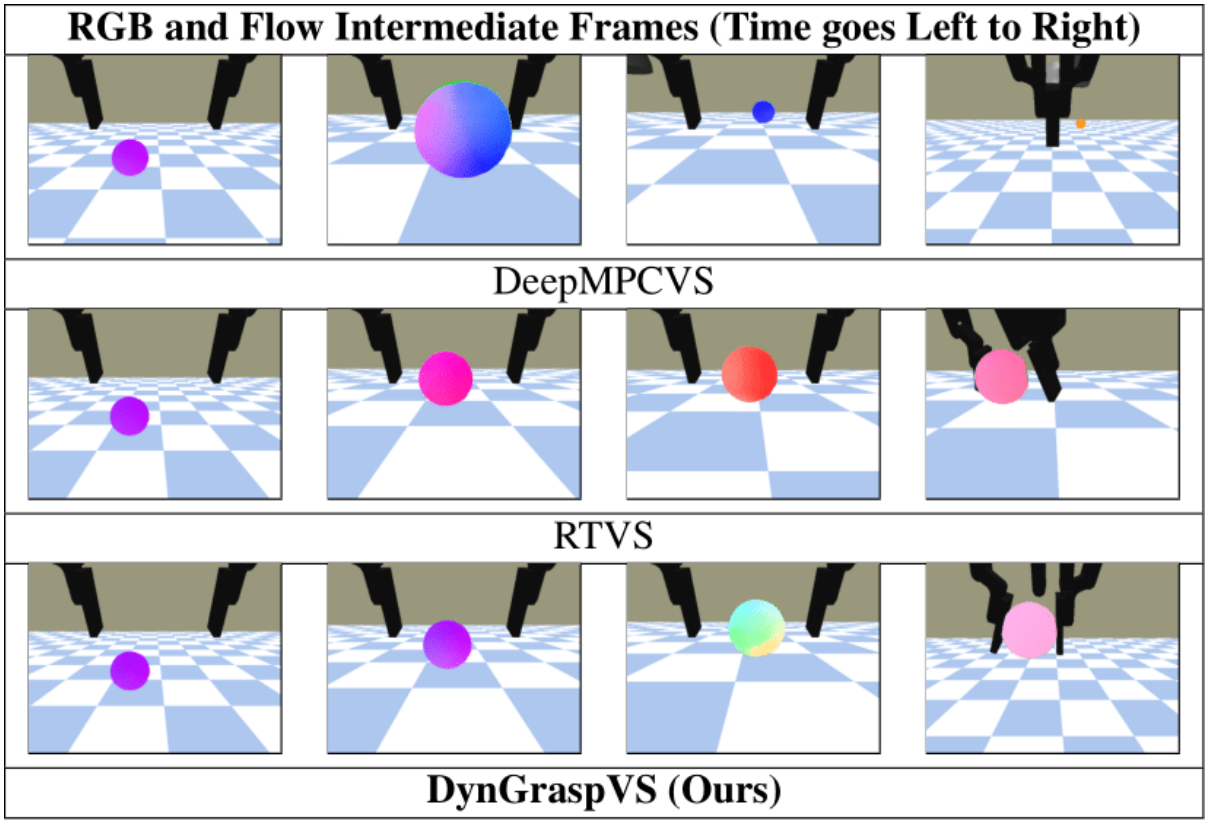


Figure 3.4: **Qualitative Results on the "Tennis ball" object of YCB dataset**: We show the RGB images overlapped with the associated Flow images for intermittent poses captured during the grasping of the Tennis ball object. Note that the flow uses the color coding mentioned in the supplementary section of FlowNet2.0 [1].

value (> 0.90) signifies successful convergence, indicative of our algorithm's adeptness in estimating the optimal interaction region with moving objects. This metric is essential for evaluating the precision and reliability of our approach in grasping tasks.

3.3.3.5 Photometric Error

We utilize photometric error as a convergence measure. It quantifies the difference between the segmented images and serves as an indicator of the algorithm's ability to maintain accurate visual alignment required for successful grasping. In our proposed approach we calculated the error on segmented image, as aids in trajectory refinement due to changing background.

3.3.4 Qualitative Results

We plot a comparison against the baselines in Fig. 3.4. It can be observed from the RGB images that the end-effector fails to to grasp the moving object (Banana) in case of RTVS and DeepMPCVS, but is successful in our case. RTVS is able to come close to the object and make a grip, however DeepMPCVS fails to make a grip. It can also be seen that the flow is lesser (color is lighter) for our case as compared to others. This can be attributed to the fact that End-effector is able follow the object much better for our case and relative position converges well.

3.3.5 Quantitative Results

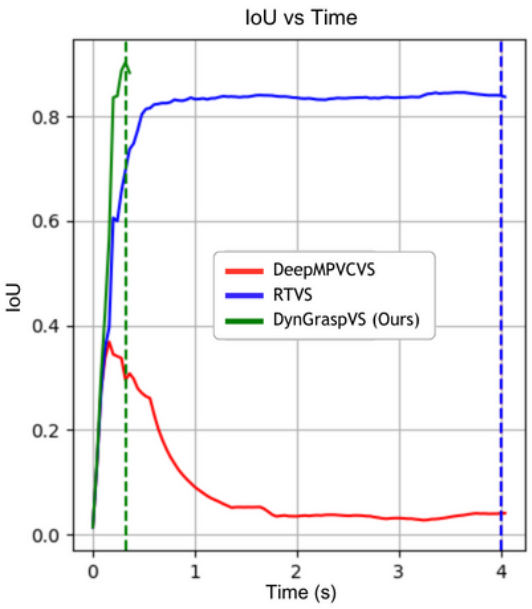
	Time ↓ (s)	**IoU** ↑	**Success** ↑ (%)	**Traj. Length** ↓ (m)
DeepMPCVS	3.95	0.11	8	1.04
RTVS	3.17	0.80	40	0.45
DynGraspVS [Ours]	**0.58**	**0.95**	**96**	**0.18**

Table 3.1: **Benchmark Comparison on YCB dataset**: The given table denotes results aggregated over all objects in YCB dataset. Our method outperforms the baselines in all metrics.

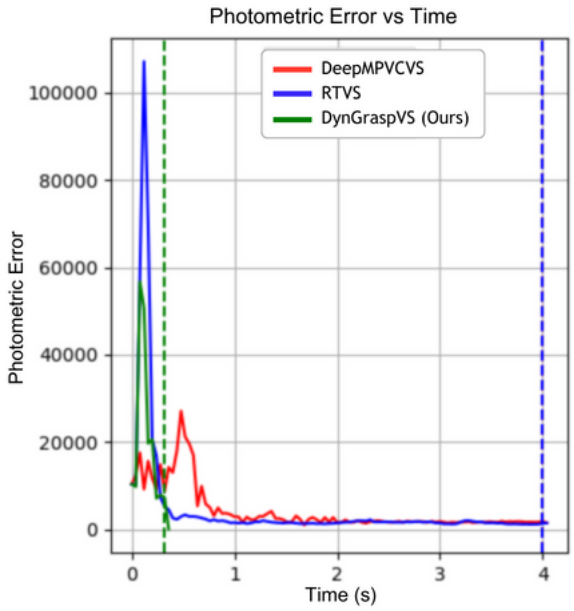
Table 3.1 presents the quantitative results on the YCB dataset. We aggregated results for reported values are averaged over multiple experiments for different objects. Our method substantially improves the baselines DeepMPCVS [33] and RTVS [9] over the different metrics.

While RTVS and DeepMPCVS have similar time taken, we achieve **5.47x** faster convergence than RTVS, and **6.81x** faster than DeepMPCVS. We also report a shorter trajectory length, reporting a **2.5x** decrease over RTVS, and **5.78x** decrease over DeepMPCVS. It proves that our method also finds an optimal minimal length trajectory to perform the grasping in comparison to others. We also report a substantially **2x** better success rate than RTVS, and **12x** over DeepMPCVS. This also proves that our method able to effectively interact and complete the grasping in presence of dynamic objects, while others fail to do so.

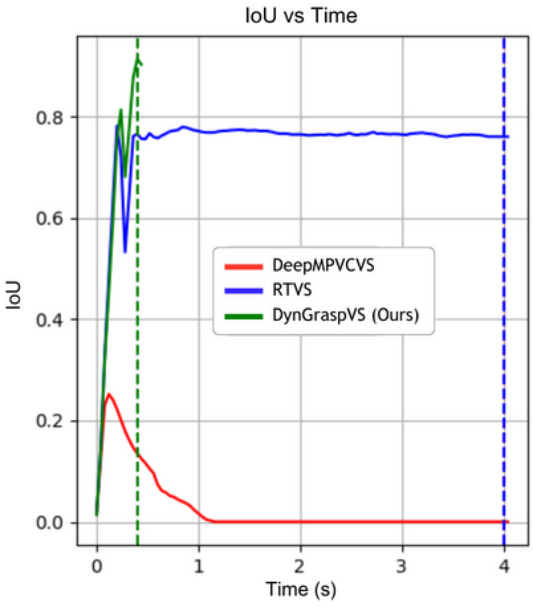
We achieve **1.18** times better IoU than RTVS, and **8.63x** better than DeepMPCVS. The IoU metric tends to be much better than both the baselines as they do not explicitly model the velocity of the object, and treat it as a stationary object.



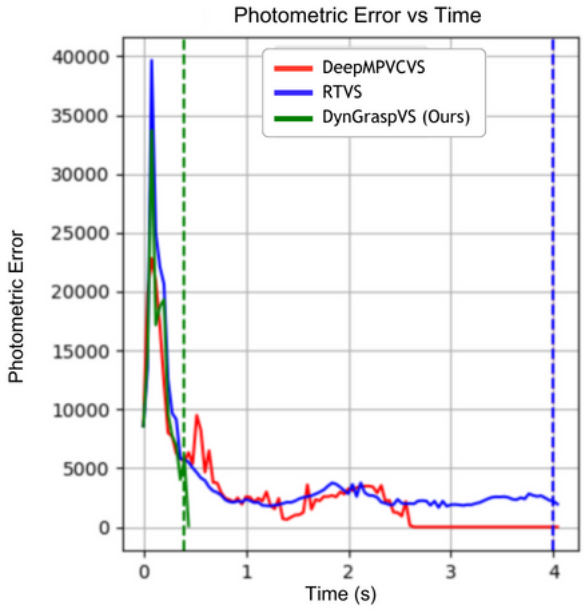
(a) Chips Can at $\vec{v}$ = (-0.09, 0.03) m/s



(b) Chips Can at $\vec{v}$ = (-0.09, 0.03) m/s

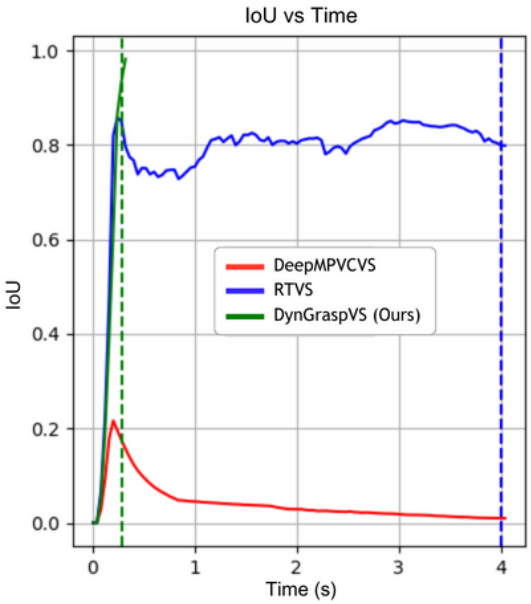


(c) Banana at $\vec{v}$ = (0.02, -0.07) m/s

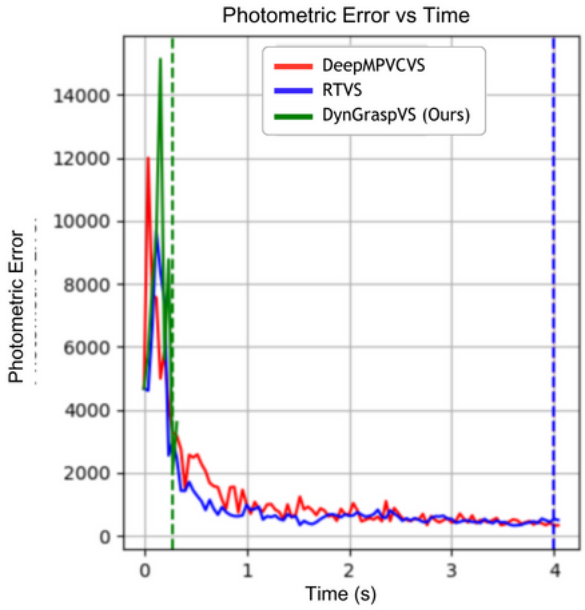


(d) Banana at $\vec{v}$ = (0.02, -0.07) m/s

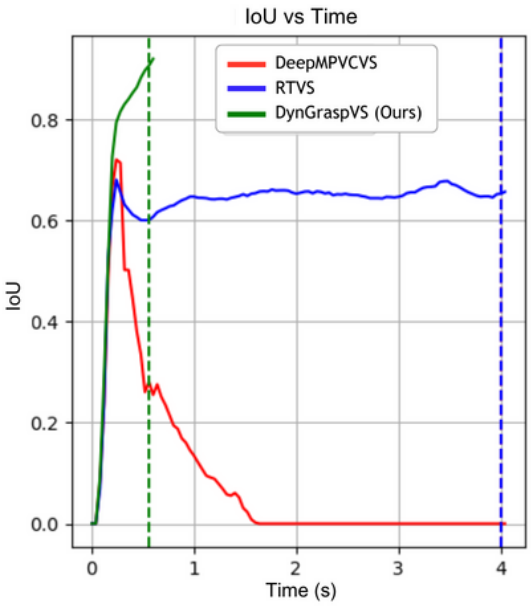
Figure 3.5: **Evolution of metrics with time for Banana and Chips Can**: Comparison of Intersection over Union (IoU) and Photometric Error over time for YCB objects Banana and Chips Can at a given velocity.



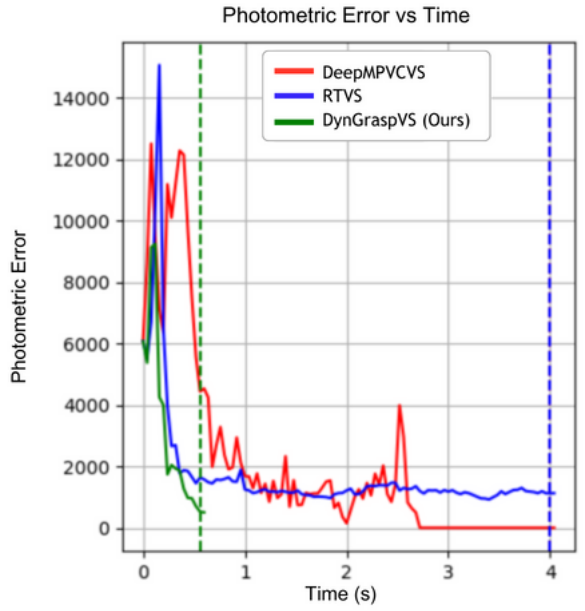
(a) Tennis Ball at $\vec{v}$ = (0.06, 0.04) m/s



(b) Tennis Ball at $\vec{v}$ = (0.06, 0.04) m/s



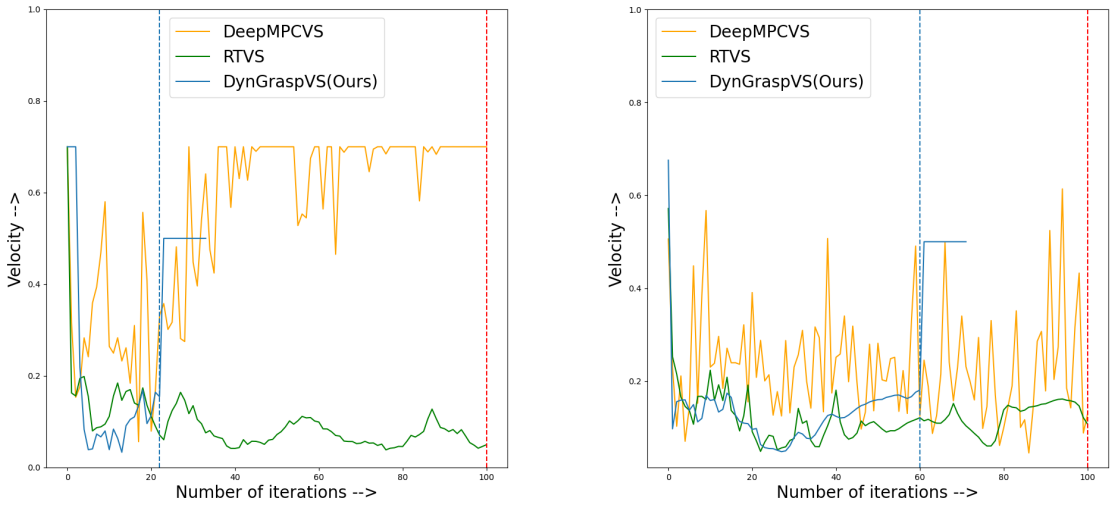
(c) Mustard Bottle at $\vec{v}$ = (-0.05, -0.05) m/s



(d) Mustard Bottle at $\vec{v}$ = (-0.05, -0.05) m/s

Figure 3.6: **Evolution of metrics with time for Mustard Bottle and Tennis Ball**: Comparison of Intersection over Union (IoU) and Photometric Error over time for YCB objects Mustard Bottle and Tennis Ball at a given velocity.

Our photometric error (100x) stays below 500 for most of the time, while the error stays high for DeepMPCVS and RTVS, indicating that our perception flow loss function is able to adapt well to the dynamic objects present in the scene. For the Chips Can example, our approach reaches convergence (photometric error falling to zero) much earlier, whereas the RTVS and DeepMPCVS approaches eventually converge at 2s. For the Mustard Bottle object, where we achieve convergence within 1s, and other approaches fail to converge even at 4s.



(a) Velocity for object moving in straight line.

(b) Velocity for object moving in circular motion.

Figure 3.7: **Velocity Comparison**: This graph shows how velocity varies with number of iterations.

In the velocity comparison depicted in fig. 3.7, our model demonstrates convergence at iteration number 22 for straight line and iteration number 60 for circular motion, as indicated by the stabilization of velocity values. A notable observation is the difference in velocity behavior between our model and the baseline approaches, DeepMPCVS and RTVS. RTVS exhibits a cautious approach characterized by smaller velocity increments, suggesting a conservative prediction of changes in velocities. Conversely, DeepMPCVS displays a tendency to take larger steps, resulting in more pronounced fluctuations and occasional overshooting in velocity predictions. This contrast highlights the distinct velocity prediction methodologies adopted by the models in question, where RTVS emphasizes a more cautious and precise adjustment strategy, whereas DeepMPCVS opts for quicker velocity changes, which might compromise precision in favor of speed.

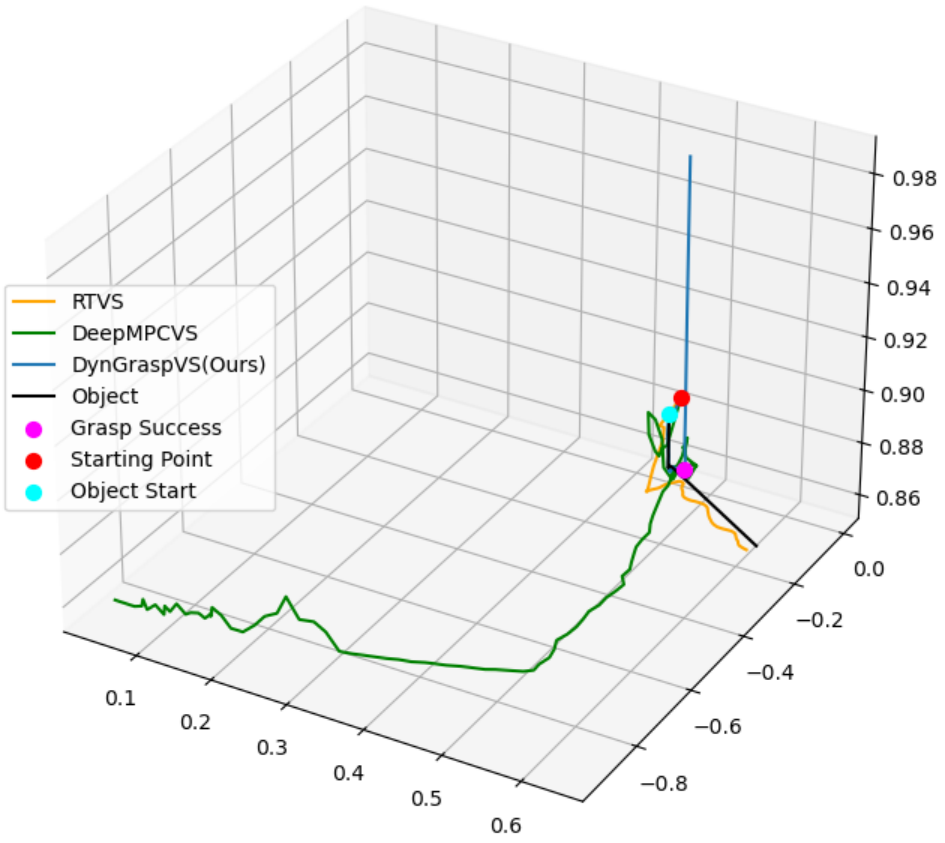


Figure 3.8: **Trajectory Comparison of object in linear motion**: We plot the trajectory of RTVS, DeepMPCVS and DynGraspVS.

In Figures, 3.8, 3.9, the trajectory depicted in black represents the path of the moving object. The presence of a pink marker indicates a successful grasp achieved by DynGraspVS, denoting the intersection point where DynGraspVS's trajectory converges with that of the object. This successful interception underscores DynGraspVS's precise timing and trajectory planning. Contrarily, the trajectory marked in green, corresponding to DeepMPCVS, diverges from the object's path, illustrating a scenario where the

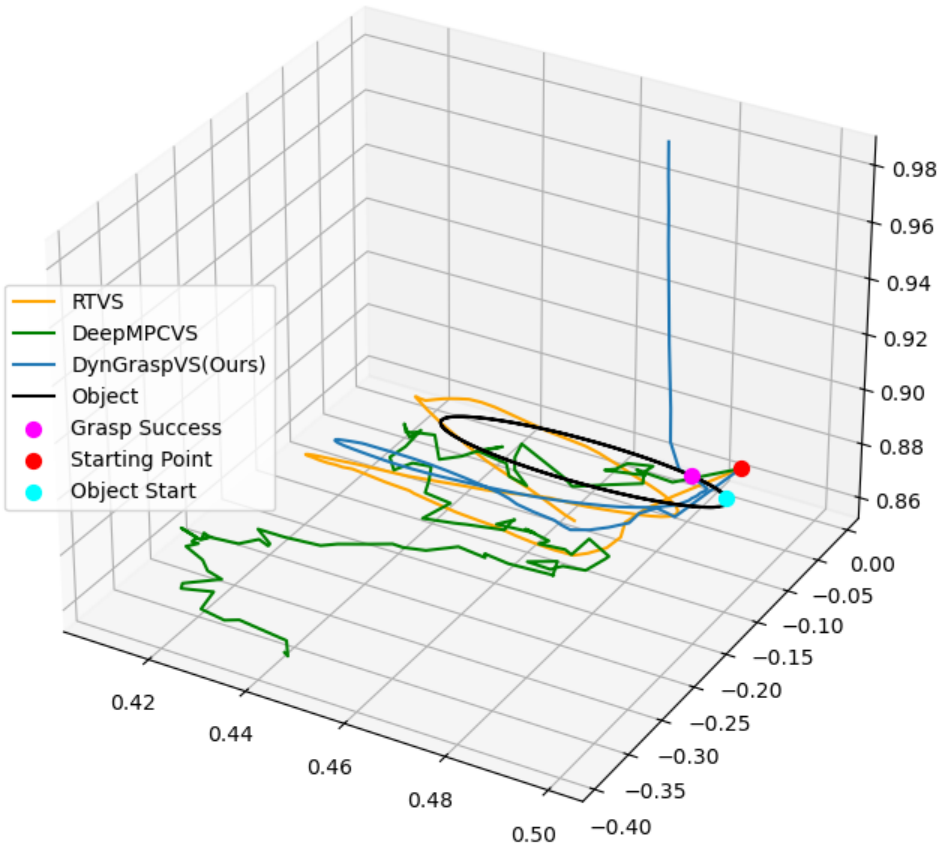


Figure 3.9: **Trajectory Comparison of object in circular motion**: We plot the trajectory of RTVS, DeepMPCVS and DynGraspVS.

robot deviates from the target rather than approaching it. The RTVS method, denoted by the orange trajectory, exhibits a highly oscillatory behavior. Despite multiple instances where it appears to near the object, RTVS consistently fails to secure a grasp, illustrating its struggle with accuracy and consistent trajectory alignment under dynamic conditions.

3.3.5.1 Object-wise Results

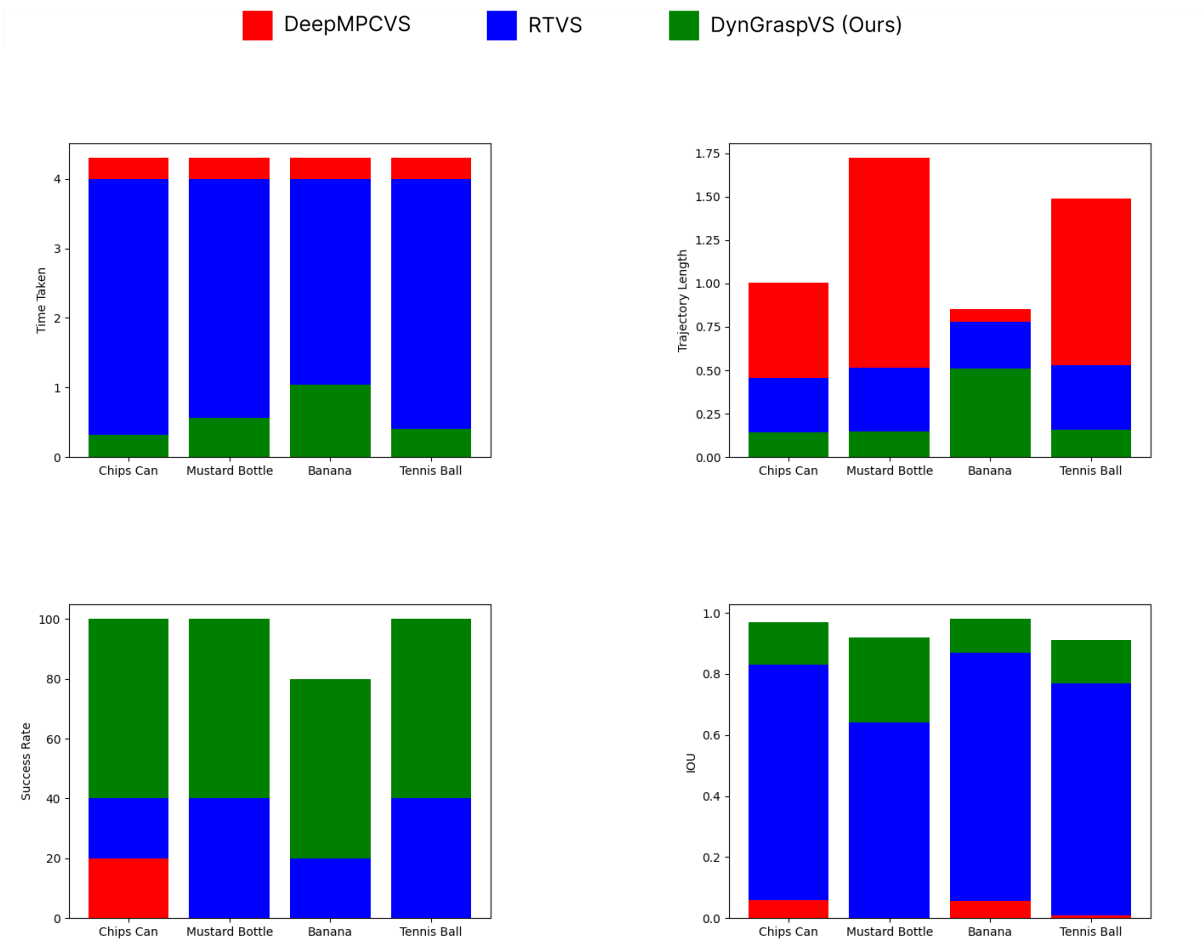


Figure 3.10: Given is the performance of our baselines on different objects in the YCB dataset. We report (a) IoU, (b) Success Rate (%), (c) Time Taken (s) and (d) Trajectory Length (m).

In Fig 3.10, we plot the metrics stated in previous section for 3 of the objects in YCB dataset - Chips Can, Mustard Bottle, Banana and Tennis Ball. We report that for Chips Can and Banana object, our approach improves upon the baselines by achieving better IoU, and Success Rate. It also achieves faster convergence than these baselines, by taking lesser time, and a shorter trajectory. For the Mustard Bottle and Tennis Ball, the improvement is substantial in terms of the trajectory length, and time taken.

Additionally, in Fig. 3.5, 3.6 we also plot IoU and photometric error with time for 4 different objects - Chips Can, Banana, Mustard Bottle and Tennis Ball with time at in Fig. 3.5, 3.6. For all the 4 objects, the IoU climbs early and reaches 0.9 within 0.5s. However, for RTVS the IoU stagnates at 0.8 for

Chips Can, 0.78 for Banana, and 0.70 for Mustard Bottle respectively and is changing for Tennis Ball around 0.8. Notably, IoU for DeepMPCVS falls after climbing for all the objects. DeepMPCVS is computationally heavy, and with the object moving away, it is unable to account for the photometric error resulting in a dip of IoU.

Chapter 4

Real-World Applications

4.1 Introduction

In this chapter, we provide an in-depth exploration of the hardware and software components of the XARM7 robotic platform in section 4.2, detailing the setup and configuration required for controlling the manipulator. We discuss the intricacies of connecting and interfacing with the XARM7 hardware and software tools utilized for commanding the manipulator's motion.

Following the setup, we look into the methodology, sec. 4.3 employed for integrating diffusion models and Recurrent Task-Visual Servoing (RTVS) into the robotic system. We explain the architecture of the proposed model. Additionally, we present a comprehensive overview of the experimental setup, including the design of tasks, data collection procedures, and performance metrics utilized for evaluating the effectiveness of the proposed approaches in section 4.4.

Subsequently, we present the experimental results obtained from conducting various tasks using the RTVS and integrated framework of RTVS and diffusion model. We analyze the performance of the system in terms of success rate of tasks and convergence in case of RTVS, providing insights into the efficacy and robustness of the proposed methodologies in real-world scenarios.

4.2 Real World Setup

This section details the hardware configuration and software setup for our UFACTORY xARM7 [59] robotic manipulator equipped with a camera for visual servoing tasks.

We establish a setup comprising the xARM 7 robotic manipulator equipped with an Intel RealSense D455 [60] camera sensor mounted on its end effector, eye-in-hand configuration. The xARM 7 is con-

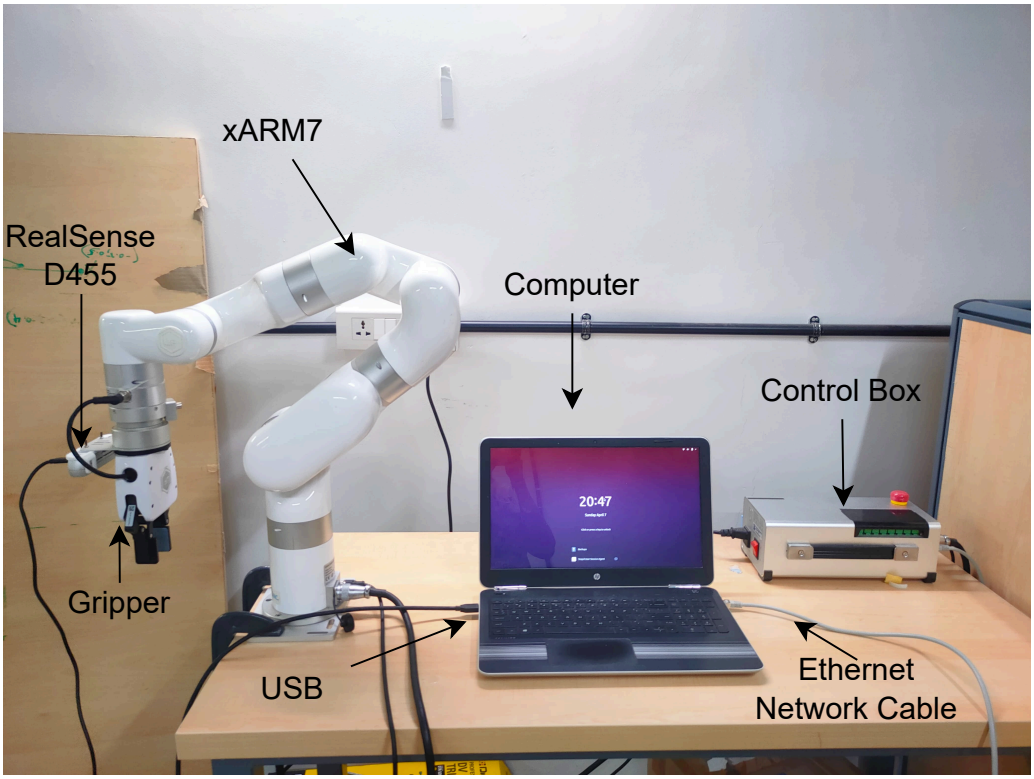


Figure 4.1: Complete Setup for working of xARM7

nected to a desktop computer via an Ethernet port, while the camera is connected to the same computer via USB. This configuration allowed for seamless communication between the manipulator, camera, and computer. The desktop computer served as the central control unit, hosting a Python script for running the entire system. This script coordinated the operation of the camera and manipulator, issuing the control commands to facilitate the execution of tasks. Once the objects were arranged within the workspace, the camera is activated to capture images at a continuous frequency of 30 Hz. Concurrently, the Python script is executed to initiate and manage the manipulator's actions, ensuring synchronized operation between the visual input from the sensor and manipulator throughout the task execution process.

4.2.1 xARM 7

The xARM7, with its 7 Degrees of Freedom and a reach of 700mm, is designed for precision and versatility in both industrial and research settings. Weighing 13.7 Kg, it supports a payload of 3.5 kg and operates with a repeatability of $\pm 0.1mm$, suitable for detailed assembly work and delicate object manipulation. Its maximum speed of 1 m/s streamlines tasks with efficiency. Equipped with advanced safety features, including collision detection and singularity avoidance, the xARM7 ensures secure operations by automatically halting motion in critical scenarios, making it a reliable tool for complex automation challenges.

The specific range of motion for each joint of the xARM7 is detailed in Table 4.1. This information is crucial for understanding the manipulator's workspace limitations and planning motion trajectories that stay within its capabilities.

Joint	**Range**
Joint 1	$\pm 360^\circ$
Joint 2	$-118^\circ \sim 120^\circ$
Joint 3	$\pm 360^\circ$
Joint 4	$-11^\circ \sim 225^\circ$
Joint 5	$\pm 360^\circ$
Joint 6	$-97^\circ \sim 180^\circ$
Joint 7	$\pm 360^\circ$

Table 4.1: xARM7 Joint Range

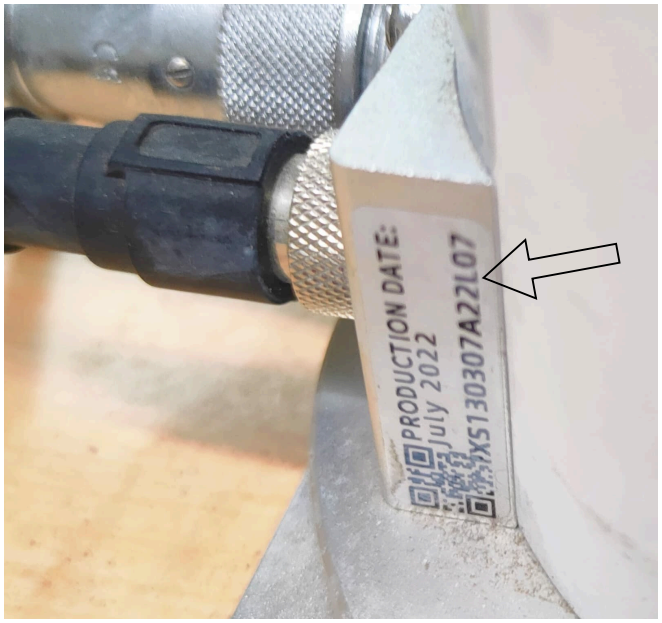


Figure 4.2: Serial Model of xARM

The third digit from the right, L, in figure 4.2 indicates that the model of the xARM7 manipulator is 2. Further details about this specific model are provided in Table 4.2, which outlines the specifications and characteristics of the xARM7 manipulator.

The xARM 7 features an integrated 2-finger gripper with a single degree of freedom, designed to perform dynamic grasping tasks. The gripper's operation range spans from -10 to 850, where a higher value corresponds to a wider grip span. This range enables precise control over the grip's tightness, allowing for adjustments until the grasp is secured. For optimal performance, the gripper's speed should be set between 1000 and 5000 to ensure the reliability in grip execution.

4.2.2 Software

For controlling the xARM 7, we employ the xARM Studio software for live configuration management via a user interface. Initial setup requires establishing a network communication link between the computer and the control box, ensuring both are on the same IPV4 network segment.

For programmatic control, the xARM-Python-SDK provided by UFactory is utilized. It necessitates specifying the control box's IP address within the code. The SDK supports two primary motion modes:

Dynamics	Mass (Kg)	Center of Mass (mm)
Link1	2.46	[0.13, 30.1, -12.0]
Link2	1.916	[0.2, -129.6, 16.9]
Link3	1.69	[46.76, -25.3, -7.46]
Link4	1.774	[70.66, -116.6, 11.7]
Link5	1.357	[-0.3, 15.6, -25.3]
Link6	1.362	[65.0, 33.4, 21.3]
Link7	0.17	[0,-6.77,-10.98]

Table 4.2: Specifications of each link

- Joint Motion: Directs the manipulator to a specified joint position at defined speed and acceleration parameters.
- Cartesian Motion: Moves the end effector to a designated Cartesian coordinate, again with predetermined speed and acceleration.

Commands issued in either mode are executed immediately without buffering, with the control box capable of processing commands at up to 250Hz. Commands sent at a frequency exceeding this threshold may be disregarded.

Additionally, three movement modes are supported.

- Position Control Mode(Mode 0): The default mode upon startup, where the control box automatically plans and executes a sequence of motion commands.
- Servo Mode(Mode 1): Designed for executing high-frequency joint position commands, this mode requires users to manually implement a trajectory planner.
- Joint Teaching Mode(Mode 2): Allows manual manipulation of the manipulator's links, provided the details like tool center point (TCP) load and payload are accurately configured.

This configuration flexibility and the integration of both manual and automated control methodologies facilitate precise manipulator control, essential for executing complex tasks and research experiments.

4.3 Methodology

Our proposed method as shown in fig. 4.3 operates through an iterative process, employing an alternating loop of intermediate goal generation and action execution to accomplish the specified task described in the language instruction. At the core of our approach lies the integration of a foresight model and a flow-based Image-Based Visual Servoing (IBVS) controller.

The foresight model* is a conditional diffusion image-to-image translation model. This model is conditioned on the current monocular eye-in-hand camera input, as well as text prompt. By leveraging this information, the foresight model generates intermediate goal images that encapsulate the desired state or configuration of the scene, guided by the task description provided.

Once the intermediate goal image is generated, it serves as the target for the flow-based IBVS controller. This controller, based on the Real Time Visual Servoing (RTVS) framework proposed by [9], utilizes optical flow features to predict and execute subsequent actions in six degrees of freedom (6 DOF) to reach the intermediate goal. This end effector position is reached by calculating the joint angles using inverse kinematics. The IBVS controller operates iteratively, continuously updating its actions based on real-time feedback from the camera input and the generated intermediate goal images.

4.3.1 Foresight Model

Unlike unconditional diffusion models that aim to learn the data distribution itself, CDMs leverage additional information to guide the generation process towards a desired outcome. In the context of image generation with textual guidance, the conditioning information is a textual prompt that describes the desired content of the image. Mathematically, a CDM can be formulated as follows: given a latent noise vector z and a conditioning variable c (text prompt in our case), the model learns the conditional diffusion process:

$$p_t(x|z,c) = \mathcal{N}\left(\mu_t(x,c), \sigma_t^2(x,c)\right),$$

where:

*Would like to acknowledge that the specific implementation details of the diffusion model are part of Pranjali Pathre's ongoing thesis work. The high-level concepts presented here provide a general understanding of how a diffusion model can be integrated into our robot control approach.

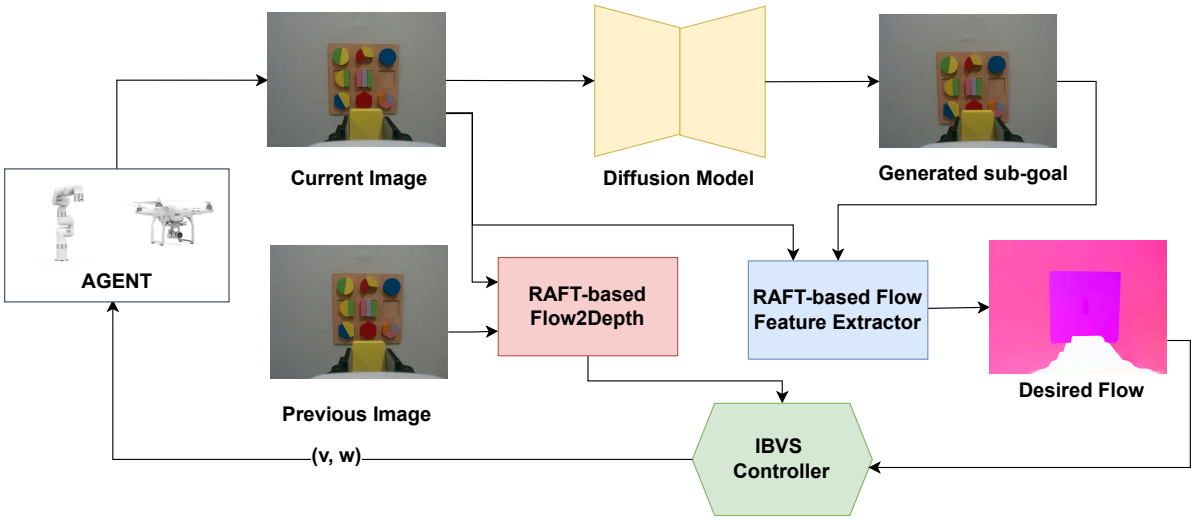


Figure 4.3: **Pipeline for Task Execution**: First, the foresight model, a diffusion-based image-to-image translation model conditioned on the current monocular eye-in-hand camera input and additional scene observations, generates the target image. Subsequently, the optical flow between the current and target image is computed to determine the motion required to reach the target. Similarly, the optical flow between the target and the previous image is utilized to predict depth using flow to depth. Once the 6 degrees of freedom (6DOF) control is predicted by the Recurrent Task-Visual Servoing (RTVS) controller, inverse kinematics (IK) is employed to calculate the joint angles necessary to move the manipulator to the predicted end effector position.

x represents the image at diffusion step t. $p_t(x|z,c)$ denotes the conditional probability density function of the image x at step t given the latent noise z and conditioning variable c. $\mu_t(x,c)$ and $\sigma_t^2(x,c)$ represent the predicted mean and variance of the image at step t, respectively, which are functions of both the current image state x and the conditioning information c.

In this work, we utilize the recent advancements in image-editing models, specifically utilizing the Instruct-Pix2Pix [61] architecture, as a 'foresight' model for generating subgoals for our servoing algorithm. The model takes as input the current image I_t from the camera sensor and outputs the subgoal image I_g.

4.3.2 RTVS Controller

In our pipeline, we integrate the RTVS [9] controller to guide the manipulator towards the desired subgoal predicted by the diffusion model. RTVS, known as Image-based Visual Servoing controller, operates within an unsupervised framework and utilizes an MPC objective to generate control commands. Unlike the loss function discussed in Section 3.2.4, RTVS's loss function does not explicitly account for moving objects. Therefore, the loss function crucial for training our real-time control generation network is distinct. It is designed to optimize the manipulator's movements based on visual feedback, enabling precise and adaptive control in real time.

$$\mathcal{L}_{flow} = \|\widehat{\mathcal{F}}(\widehat{\mathbf{v}}_t) - \mathcal{F}(I_t, I^*)\| = \|[L(Z_t)\widehat{\mathbf{v_t}}] - \mathcal{F}(I_t, I^*)\| \tag{4.1}$$

In RTVS, $\mathcal{F}(I_t, I^*)$ is the desired optical flow between the current image, I_t and the sub-goal image, $I*$ and $L(Z_t)\widehat{\mathbf{v_t}}$ is an estimation of the movement between the current state and the desired subgoal. The interaction matrix, $L(Z_t)$ is described in eq. 4.2.

$$L(Z_t) = \begin{bmatrix} -1/Z_t & 0 & x/Z_t & xy & -(1+x^2) & y \\ 0 & -1/Z_t & y/Z_t & 1+y^2 & -xy & -x \end{bmatrix} \tag{4.2}$$

This MPC objective guides the network towards generating control signals that effectively minimize this discrepancy, thus steering the robot towards the desired outcome.

4.3.3 Inverse Kinematics

As RTVS provides us with the with the predicted end-effector velocity ($V_e = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]$) at each step, we now need to move the end-effector to a specific position and orientation. This necessitates solving the inverse kinematics (IK) problem, which involves determining the joint configurations

(angles) needed to reach a desired end-effector pose. For a 7 DOF manipulator, this task can be computationally challenging.

In our work, we addressed this challenge by leveraging the Jacobian method, a well-established numerical approach for IK. The Jacobian method allows us to translate this desired end-effector motion into the corresponding joint velocities required to achieve it. The core principle lies in the relationship between joint velocities ($\dot{q} = [\dot{q}_1, \dot{q}_2..., \dot{q}_7]$) and the end-effector twist (combination of linear and angular velocities) as expressed by the Jacobian matrix, J(q):

$$V_e = J(q).\dot{q} \tag{4.3}$$

The Jacobian matrix can be expressed as

$$J(q) = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \cdots & \frac{\partial x}{\partial q_7} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \cdots & \frac{\partial y}{\partial q_7} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \cdots & \frac{\partial z}{\partial q_7} \\ \frac{\partial r}{\partial q_1} & \frac{\partial r}{\partial q_2} & \cdots & \frac{\partial r}{\partial q_7} \\ \frac{\partial p}{\partial q_1} & \frac{\partial p}{\partial q_2} & \cdots & \frac{\partial p}{\partial q_7} \\ \frac{\partial \gamma}{\partial q_1} & \frac{\partial \gamma}{\partial q_2} & \cdots & \frac{\partial \gamma}{\partial q_7} \end{bmatrix} \tag{4.4}$$

Here, J(q) is a 6x7 matrix that depends on the end effector position (x,y,z,r,p,γ) and the current joint angles (q). The elements of this matrix capture how each joint's motion contributes to the end-effector's overall linear and angular velocities.

The end effector position can also be calculated using the manipulator's Denavit-Hartenberg (DH) parameters, which provide a systematic method for determining the kinematics of robotic manipulators. These parameters define the geometric relationship between adjacent links in the manipulator and allow for the computation of the end effector position. For our XARM7 manipulator, we utilize the DH parameters listed in Table 4.3.

The DH parameters consist of four parameters, a_i, α_i, d_i, θ_i, where i represents the index of the joint. These parameters describe the link lengths, link twist angles, link offsets, and joint angles, respectively. By applying the DH transformation matrices recursively from the base to the end effector, we can determine the position and orientation of the end effector relative to the base frame.

Joint	θ (rad)	d (mm)	a (mm)	α (rad)
Joint 1	0	267	$-\pi/2$	0
Joint 2	0	0	$\pi/2$	0
Joint 3	0	293	$\pi/2$	52.5
Joint 4	0	0	$\pi/2$	77.5
Joint 5	0	342.5	$\pi/2$	0
Joint 6	0	0	$-\pi/2$	76
Joint 7	0	97	0	0

Table 4.3: Standard D-H Parameters of xARM7

While directly solving for joint angles might be challenging, the Jacobian allows us to compute the required joint velocities for a desired end-effector twist using eq. 4.5

$$\dot{q} = J(q)^{-1} V_e \tag{4.5}$$

Since, the J(q) is a non-square matrix, we utilize the pseudo-inverse ($J(q)^+$) of the Jacobian matrix to compute the joint velocities required to achieve a desired end effector velocity.

$$\dot{q} = J(q)^{+} V_e \tag{4.6}$$

The pseudo-inverse allows us to approximate the inverse of a non-square matrix and is calculated using the Singular Value Decomposition (SVD) method.

However, it's important to acknowledge that the Jacobian might not always have a unique inverse, particularly at singular configurations where the manipulator loses dexterity. In such cases, alternative approaches or optimization techniques are used to avoid such cases.

For practical implementation, we leveraged the capabilities of PyBullet, a popular physics simulation library. PyBullet provides a convenient function, calculateInverseKinematics, that simplifies the IK computation. This function takes the desired end-effector pose, the current joint configuration, and the robot model as inputs and returns the corresponding joint angles. By utilizing PyBullet's efficient implementation, we were able to effectively solve the IK problem for our 7 DOF manipulator and achieve precise control of its end-effector throughout our experiments.

4.4 Experiments and Results

We plan and conduct our experiments within the workspace of the manipulator, ensuring that all tasks remain feasible within the operational boundaries of the robotic system. The workspace of the manipulator is defined by the range of motion it can achieve in the x, y, and z dimensions, relative to the base link position of the xARM. Specifically, the workspace spans from $x = 0.255$ m to $x = 0.568$ m, $y = -0.098$ m to $y = 0.22$ m, and $z = 0.48$ m to $z = 0.636$ m. These boundaries outline the region within which the manipulator can effectively perform tasks and interact with objects, guiding our experimental setup and ensuring that all planned actions remain within feasible operational limits. Moreover, we use the xARM Python SDK in mode 0 to execute joint motions, facilitating precise and controlled movements during our experiments.

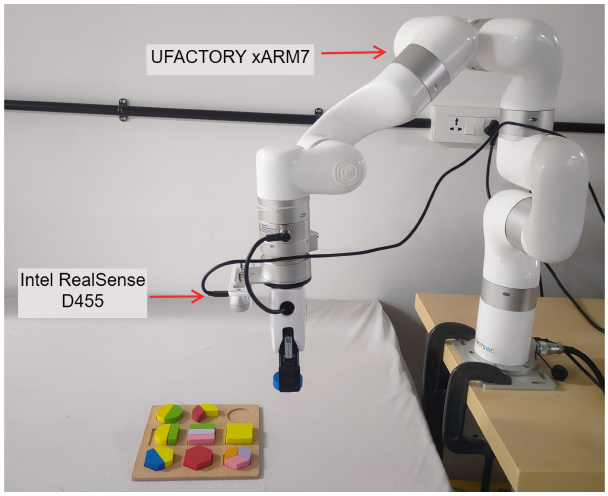


Figure 4.4: **Real-world setup**: We employed a UFACTORY xARM7 robotic manipulator equipped with an eye-in-hand Intel RealSense D455 depth camera.

4.4.1 Dataset

We designed a data collection strategy that involved generating a significant amount of diverse training examples. For each task our system was designed to handle, we generated 400 unique trajectories. Each trajectory captured a sequence of 15 video frames. This approach ensured the model was exposed to a variety of potential motions and scenarios within each task context.

We sampled input-output image tuples from existing video demonstrations showcasing the desired manipulator actions for each task. An input image represents object configuration while that step and

the output image depicts the desired outcome of the manipulator's manipulation. These image tuples were paired with language prompts that explicitly explained the task objective.

4.4.2 Evaluating RTVS Efficacy in Real World

In our initial real-world testing of the RTVS (Real-Time Visual Servoing) system, we opted for a straightforward scenario to assess its convergence capabilities. The experiment involved a basic setup where a box was positioned on the floor. Our primary objective was to evaluate whether the RTVS framework could effectively converge in a real-world environment. By employing this simplified scenario, we aimed to gauge the system's ability to perceive visual cues, generate appropriate control actions, and navigate towards the target object successfully. This preliminary test served as a crucial step in validating the functionality and efficacy of the RTVS system in practical settings, laying the groundwork for more complex experiments and applications in the future.

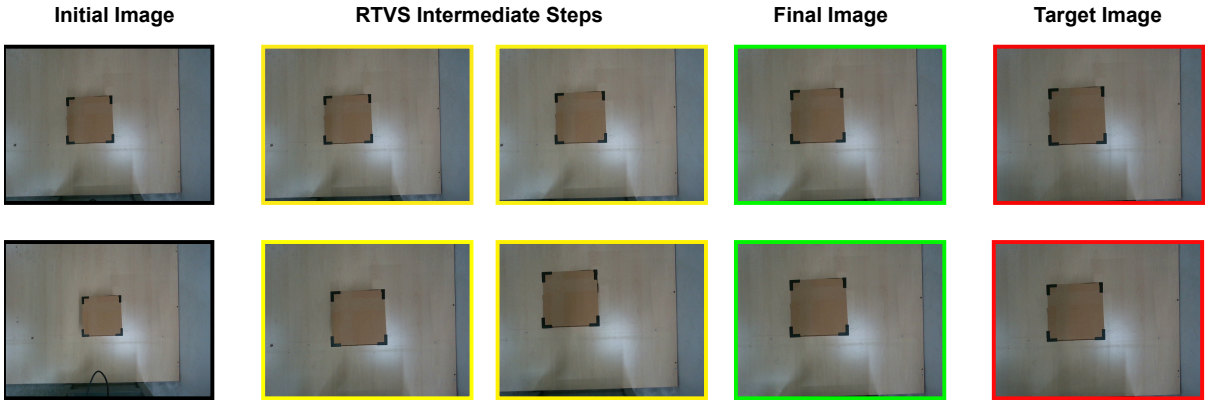


Figure 4.5: **Converge Results**: This figures shows that the RTVS is able to converge in the basic setup where there is object on the simple floor.

The RTVS demonstrates its convergence capability with a photometric error of 450. In our visual representation of the convergence process, the initial images are bordered in black, while the intermediate steps are highlighted in yellow. The final reached image is marked in green, representing the achieved state, while the target or goal image is depicted in red, indicating the desired outcome. In the first scenario, the manipulator requires movement solely in the z-direction, achieving convergence in just 5 steps. Contrastingly, in the second scenario, the initial end-effector position is 10 cm up and 5 cm left from the final position. Here, the convergence process takes longer, spanning 56 steps to reach the desired outcome.

Following our successful validation of RTVS in simpler scenarios, we introduce more complexity to the environment to assess its performance in navigating intricate surroundings and accomplishing specific tasks. In these advanced scenarios, RTVS is tasked with reaching designated positions within complex environments.

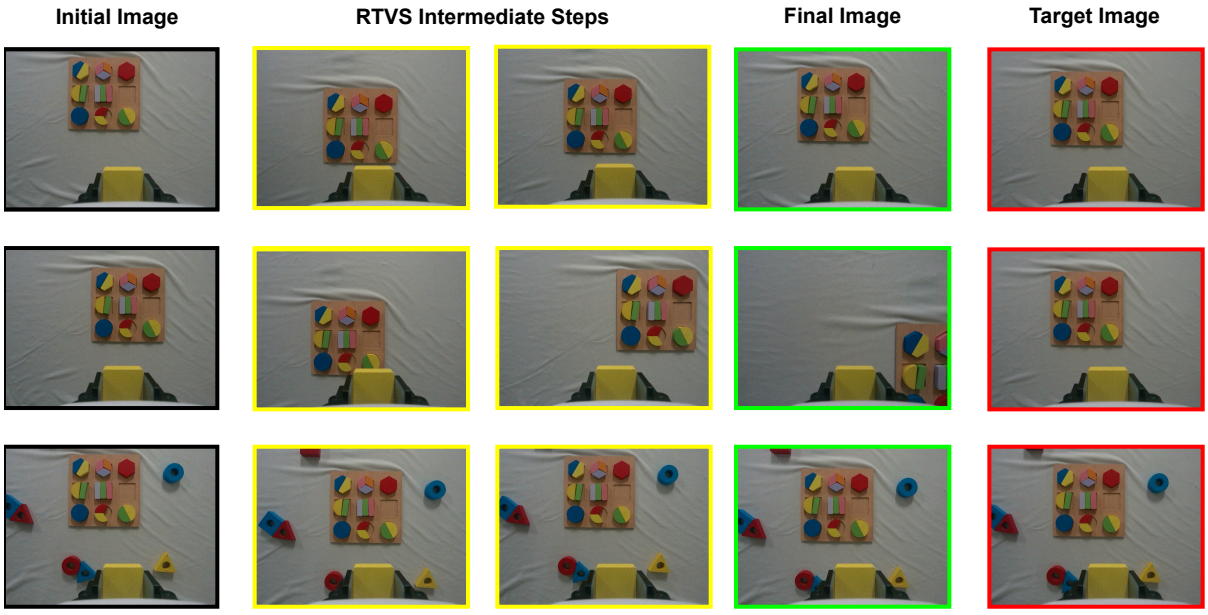


Figure 4.6: **Converge Results**: The RTVS results on a complicated environment.

In Figure 4.6, we observe two trajectories wherein RTVS exhibits divergent behaviors under different initial conditions despite targeting the same destination. In trajectory 1, RTVS successfully converges with a photometric error of 800, indicating effective task completion. However, in the second trajectory, despite aiming for the same target position, the manipulator fails to reach the desired location within 1000 steps, rendering it unsuitable for real-world applications. To address this, we explore enhancements by augmenting the feature set to improve optical flow, a crucial component for RTVS performance. Increasing the number of features leads to a modest improvement in convergence rates, rising from 40% to 74%, yet falling short of desired efficacy. Subsequently, we investigate the impact of background variations on RTVS performance, as depicted in Figure 4.7, employing diverse backgrounds to discern their effect on convergence rates and overall system robustness.

The experimentation with various backgrounds, as illustrated in Figure 4.7, shed light on the limitations of Flownet2's capacity to accurately predict optical flow in complex real-world scenarios. Specifically, the checkerboard background produced notably chaotic flow predictions. This disruption arises

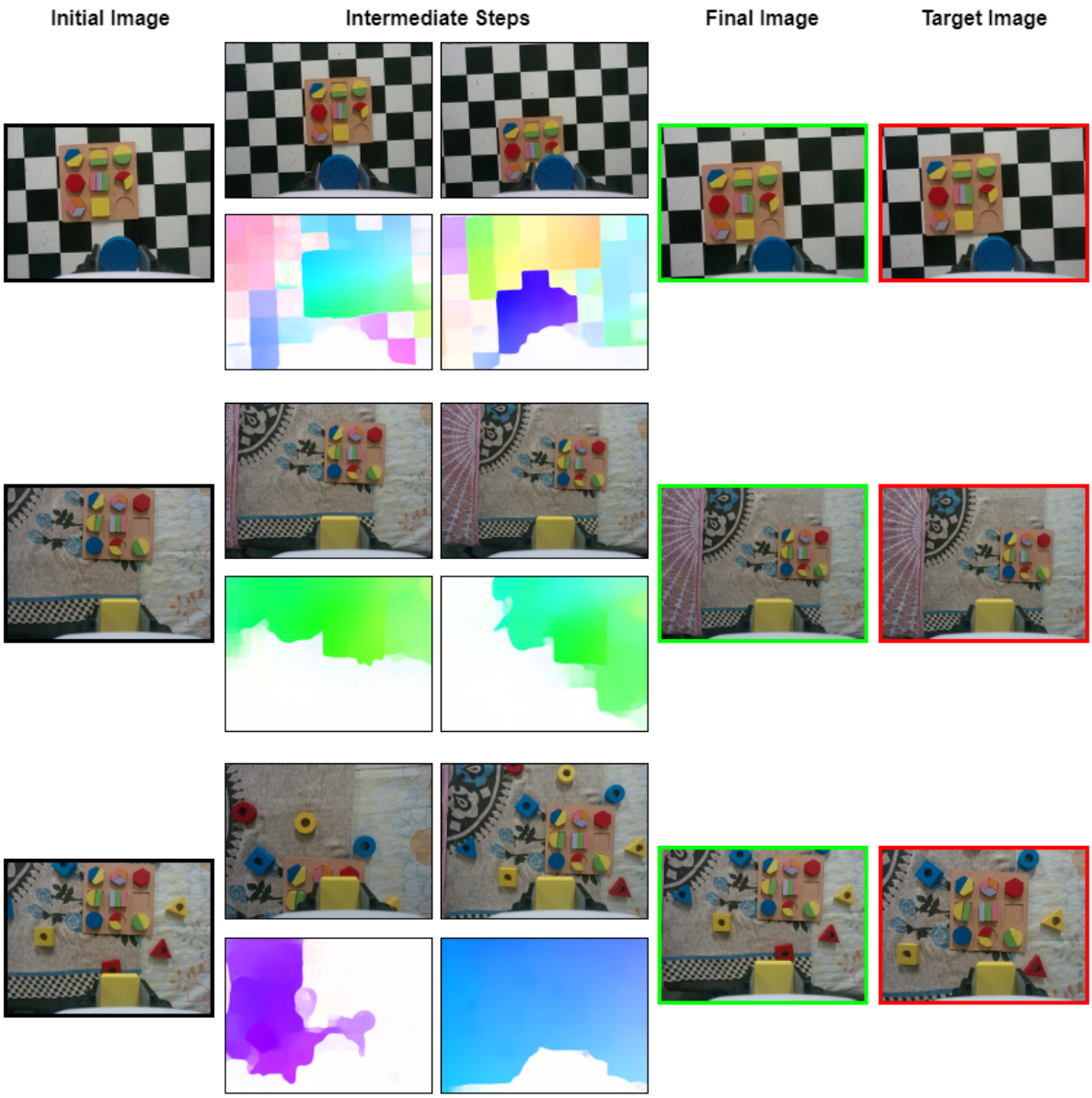


Figure 4.7: **Comparison of various backgrounds**: The RTVS results variations of backgrounds.

from flownet's challenges in consistently matching the cube from its initial position to its final one against the visually intricate checkerboard pattern. Similarly, experiments conducted against a bed-sheets background predominantly resulted in white optical flow fields. This outcome indicates a failure in capturing correct velocities, significantly attributed to the background's lack of distinguishable features. Although the introduction of additional features offered some improvement, it was insufficient to counteract the camouflage effect, where objects blend into the background, further complicating the

identification process. This series of tests underlines the critical need for adaptive or enhanced feature detection methods in complex environments to ensure the reliability and effectiveness of visual servoing systems like RTVS in real-world applications.

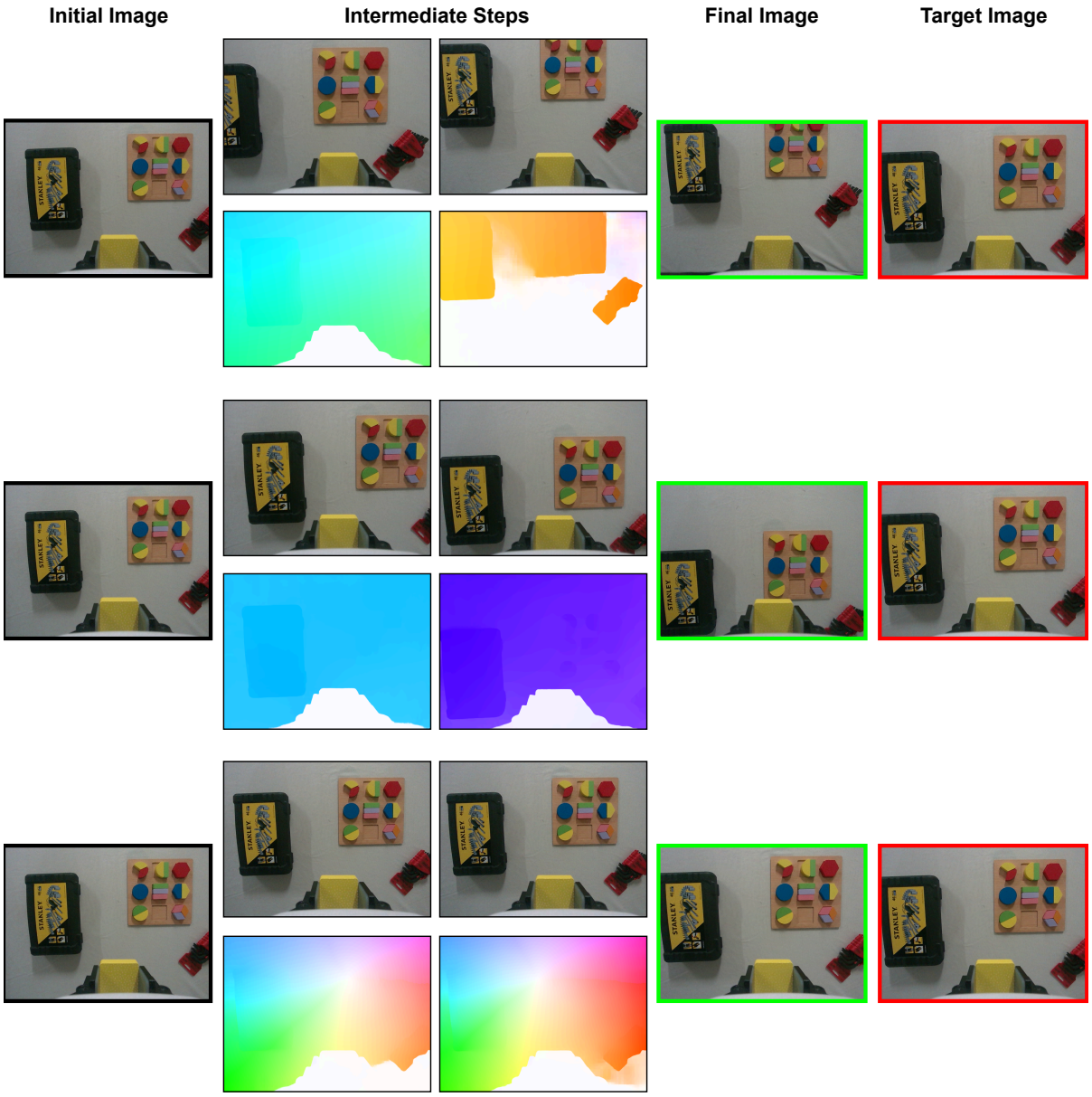


Figure 4.8: **Flow Comparison**: This figure shows the comparison between various optical flow methods.

The comparative analysis of trajectories in fig. 4.8 highlights the critical dependency of robotic visual servoing systems on the accuracy of optical flow and depth perception technologies. In trajectory

1, the reliance on Flownet2 for optical flow estimation resulted in convergence issues, primarily due to inadequate flow prediction quality. This challenge was vividly demonstrated when transitioning to trajectory 2, where, despite switching to RAFT [62], a decision motivated by Flownet's inconsistent performance—convergence was still not achieved. The persistent issue in trajectory 2 was traced back to the noisy depth data provided by the RealSense D455 camera, which underscored the sensitivity of visual servoing systems to precise depth information.

The pivotal advancement in our study was achieved with trajectory 3, marked by the successful convergence facilitated by the integration of Normalised FlowDepth explained in [9]. The transition to Two view depth normalisation addressed previous challenges related to velocity prediction errors, which were primarily due to unreliable flow estimations. The effectiveness of FlowDepth was previously compromised by inaccuracies in flow prediction; however, the incorporation of RAFT for more dependable flow estimation enabled the productive use of Normalised FlowDepth. This adaptation was crucial for overcoming the limitations encountered with earlier approaches, showcasing the critical role of precise flow estimation coupled with depth data in improving the performance of visual servoing systems.

This progression underscores a pivotal finding: the effectiveness of visual servoing in robotic systems is profoundly influenced by the interplay between optical flow accuracy and depth perception fidelity. Our experiments demonstrate that enhancements in flow prediction, when coupled with innovative approaches to handling depth data, can significantly improve the performance of visual servoing systems. This insight directs future research towards optimizing both flow estimation methods and depth data handling to achieve reliable and efficient robotic navigation and manipulation in complex environments.

Figure 4.9: Trajectory demonstrating convergence utilizing flow depth and RAFT.

This figure 4.9 shows the trajectory that achieves convergence with less than 200 steps by employing two-view normalised depth alongside RAFT.

To conclude, after extensive experimentation, it was found that the RTVS system performed effectively when integrated with two-view normalized depth strategies, specifically normalized flow depth

and RAFT. This combination proved to be successful in enhancing the system's ability to accurately interpret real-world environment and interacting with it.

4.4.3 Results

We demonstrate our diffusion model-generated sub-goals in conjunction with the RTVS controller to accomplish two distinct tasks:

- **Task 1**: Place the shape in the shape sorter.
- **Task 2**: Stack the shape into a designated sorter slot.

The tasks of sorting and stacking, as illustrated in Figures 4.10 and 4.11, serve as compelling demonstrations of the sophisticated capabilities enabled by our integration of the Imagine2Servo model and the RTVS system within a real-world robotic application. The diffusion model, a cornerstone of our approach, plays a pivotal role in generating visually coherent and contextually relevant sub-goal images, which are foundational to guiding the robot's actions toward task completion.

In Task 1 (Sorting), the efficacy of our method is showcased through its ability to discern and classify objects based on shape, subsequently manipulating them into their correct locations with an 80% success rate. This task not only tests the precision of the robot's movement and control but also the model's capacity to understand and interpret complex visual scenes. The blue images represent the model's predictions or sub-goals, which are impressively close to the real-world scenarios depicted by the yellow images. These final images represent the successful state achieved after RTVS execution, indicating the system's high level of accuracy and the practical applicability of our method in sorting tasks.

Task 2 (Stacking) elevates the challenge by requiring precise alignment for the stacking of objects. This task is significantly more demanding due to the need for exact positioning to ensure that objects are not only correctly identified and picked up but also placed with a high degree of accuracy. The success rate of 70% in this task underscores the robustness of our approach, particularly in handling tasks that require delicate manipulation and spatial awareness. The ability of the foresight model to generate actionable sub-goals, even with partially visible objects, highlights the advanced perceptual capabilities of our system. It navigates the complexities of the physical world, where visibility and positioning play crucial roles in task execution.

Crucially, both tasks demonstrate the system's resilience to real-world operational challenges, such as actuation noise, which can significantly impact the precision and reliability of robotic systems. Despite

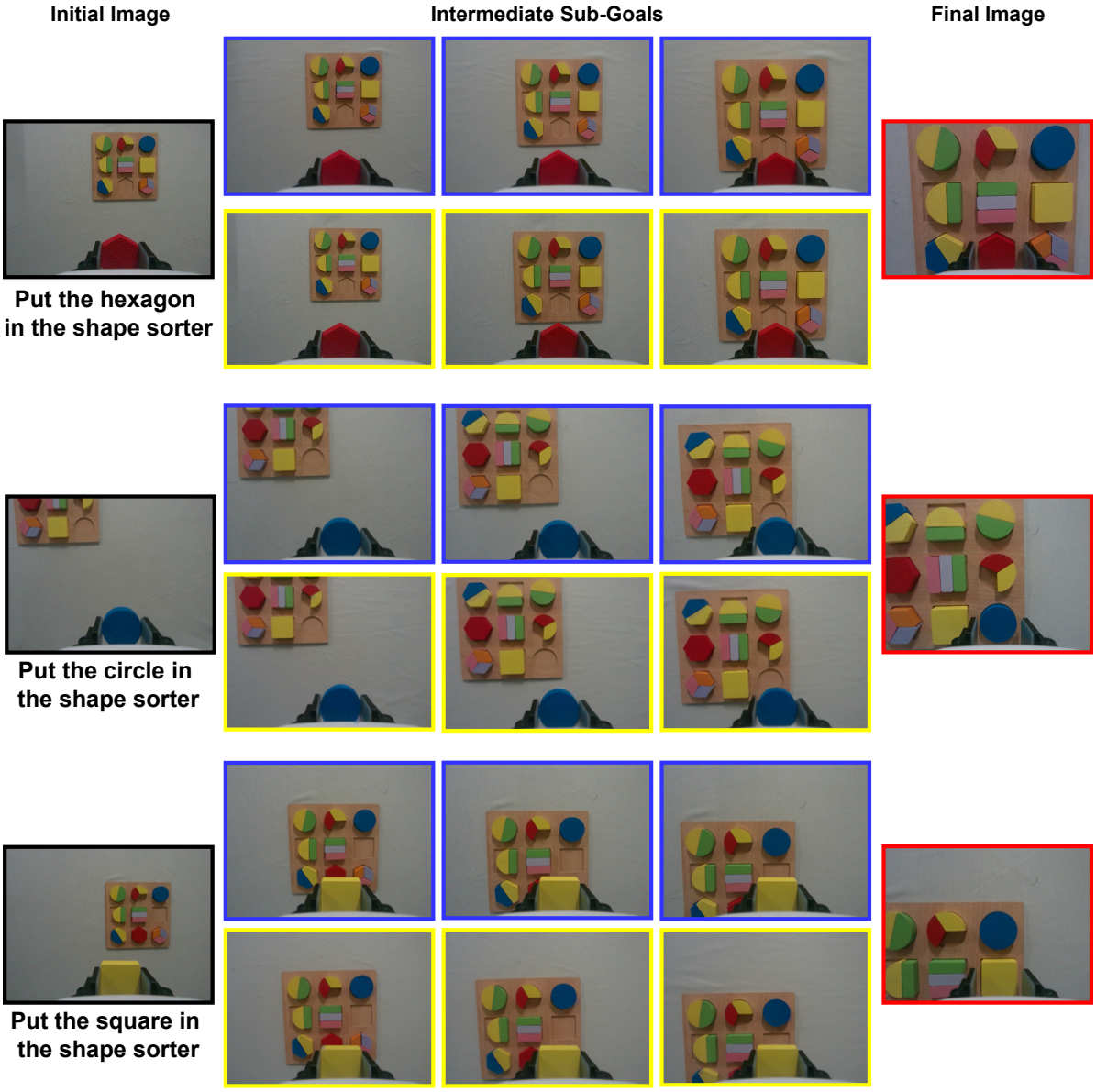


Figure 4.10: Qualitative Analysis of Task 1.

these potential disturbances, our method exhibits remarkable adaptability and accuracy. It consistently converges towards the desired outcomes, maintaining a photometric error within an acceptable range across various scenarios. This level of performance attests to the potential of integrating advanced visual servoing techniques with diffusion models in robotic applications, promising enhancements in autonomy, efficiency, and applicability across a broad spectrum of tasks in dynamic and unstructured environments.

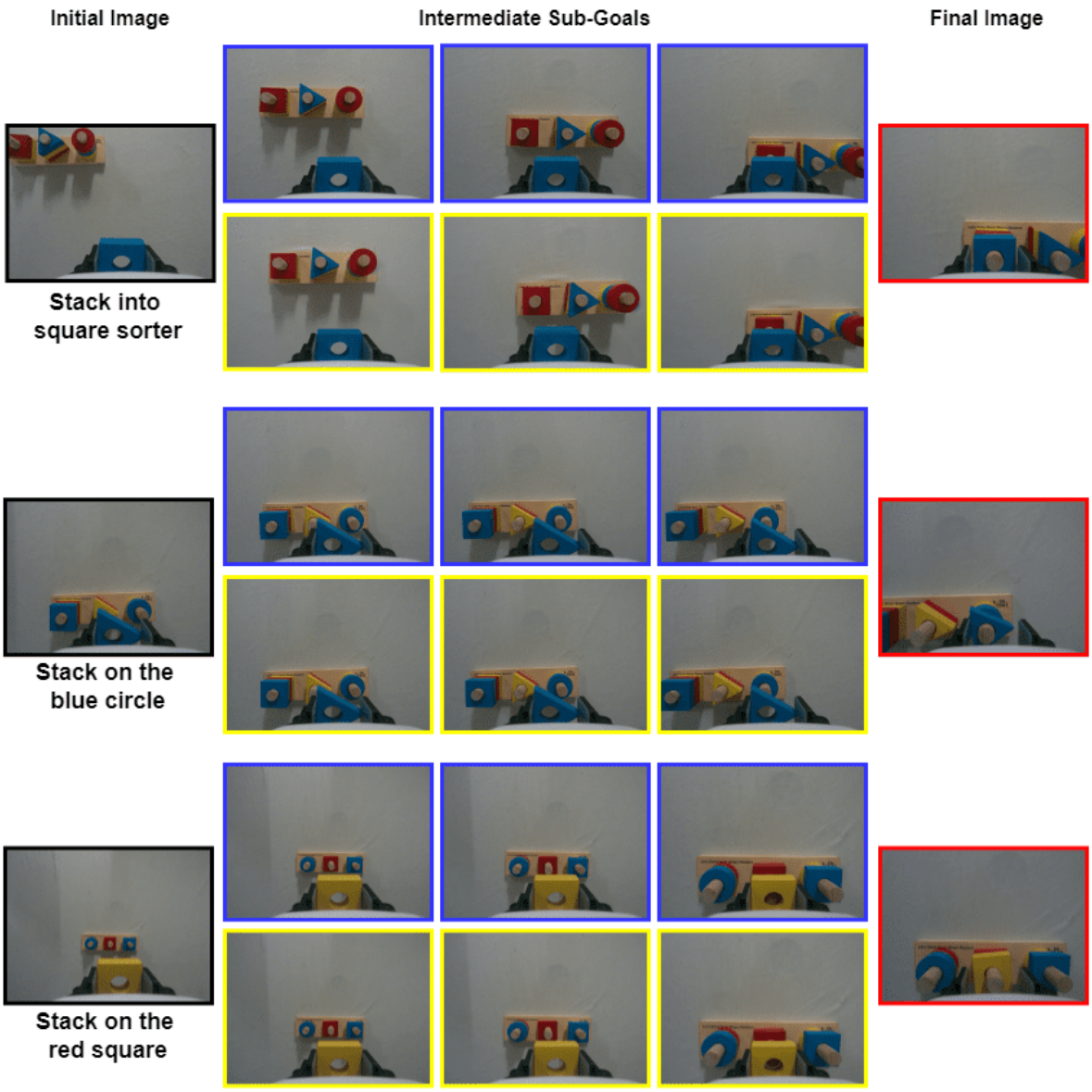


Figure 4.11: Qualitative Analysis of Task 2.

4.5 Challenges

The real-world implementation of our model encountered several challenges, each impacting the overall performance and reliability of the visual servoing tasks. One significant issue was the noisy depth information obtained from the RealSense D455 camera. Depth data, crucial for accurately positioning and orienting the robotic arm in three-dimensional space, was compromised by noise, making precise control more challenging and leading to less reliable task execution.

Lighting conditions posed another considerable challenge. Ideally, consistent lighting is necessary for visual servoing systems to accurately recognize and track objects. However, in our setup, either excessive lighting led to the creation of shadows, introducing variability in each image captured, or inadequate lighting reduced the system's ability to discern objects. Shadows, in particular, created inconsistencies in the visual data, complicating the task of image matching and affecting the algorithm's ability to converge accurately on the target.

Furthermore, the physical interaction between the robotic gripper and the objects introduced additional complexities. Achieving the delicate balance of applying sufficient force to grip objects securely without causing them to stick to the gripper proved difficult. Gripping objects with too much force resulted in them sticking to the gripper, complicating the release process and, in some cases, affecting the task's success. Conversely, gripping too lightly risked the object slipping or not being secured properly, also leading to task failure. However, solving this issue is out of scope of this thesis.

These challenges highlight the intricacies and unpredictability of translating visual servoing systems from controlled environments to the real world.

Chapter 5

Conclusions

In conclusion, this thesis has made significant strides in advancing the field of visual servoing technology, particularly in the context of real-world robotic applications. Through the introduction of DynGraspVS, a novel Visual Servoing-aided Grasping approach, we have addressed the challenges associated with grasping in dynamic environments, achieving remarkable success rates and trajectory convergence while maintaining precise alignments. By leveraging a single-step rollout strategy and integrating velocity information into the interaction matrix, DynGraspVS outperforms existing MPC-based methods, demonstrating its efficacy in complex grasping tasks.

Furthermore, our exploration of IBVS mechanisms on the XARM7 robotic platform has demonstrated the practical applicability and feasibility of IBVS in real-world scenarios. Through successful integration and implementation, we have shown how IBVS can enhance the capabilities of robotic systems, enabling them to perform tasks with greater flexibility and adaptability. Additionally, our analysis of the RTVS framework's performance in diverse real-world scenarios has provided valuable insights into its robustness and versatility, highlighting its potential for various applications. Moreover, the utilization of Imagine2Servo, a conditional diffusion model for generating target images, has further enhanced the capabilities of IBVS, particularly for complex tasks requiring precise visual feedback. Through a combination of experimental validation and rigorous testing, we have demonstrated the effectiveness of IBVS in real-world robotic systems, setting the stage for future advancements in visual servoing technology.

Related Publications

1. **Gunjan Gupta**, Vedansh Mittal, and K. Madhava Krishna. "Dyngraspvs: Servoing aided grasping for dynamic environments", in *2023 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1–8, 2023.

2. Harshit K. Sankhla, M. Nomaan Qureshi, Shankara Narayanan V., Vedansh Mittal, **Gunjan Gupta**, Harit Pandya, and K. Madhava Krishna. "Flow synthesis based visual servoing frameworks for monocular obstacle avoidance amidst high-rises", in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 1339–1345, 2022.

3. Pranjali Pathre, **Gunjan Gupta**, M. Nomaan Qureshi, Mandyam Brunda, Samarth Brahmbhatt and K. Madhava Krishna. "Imagine2Servo: Intelligent Visual Servoing with Diffusion-Driven Goal Generation for Robotic Tasks", in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024. [Submitted]

Bibliography

[1] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks, 2016.

[2] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation*, 8(3):313–326, 1992.

[3] Peter I Corke. Visual control of robot manipulators–a review. *Visual Servoing: Real-Time Control of Robot Manipulators Based on Visual Sensory Feedback*, pages 1–31, 1993.

[4] V. Kalaichelvi Abhilasha Singh and R. Karthikeyan. A survey on vision guided robotic systems with intelligent control strategies for autonomous tasks. *Cogent Engineering*, 9(1):2050020, 2022.

[5] Xiang Li, Weiwei Shang, and Shuang Cong. Model-based reinforcement learning for robot control. In *2020 5th International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 300–305, 2020.

[6] Guoguang Du, Kai Wang, Shiguo Lian, and Kaiyong Zhao. Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review. *Artificial Intelligence Review*, 54(3):1677–1734, August 2020.

[7] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes, 2021.

[8] Reuven Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

[9] M. Nomaan Qureshi, Pushkal Katara, Abhinav Gupta, Harit Pandya, Y V S Harish, AadilMehdi Sanchawala, Gourav Kumar, Brojeshwar Bhowmick, and K. Madhava Krishna. Rtvs: A

lightweight differentiable mpc framework for real-time visual servoing. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3798–3805, 2021.

[10] M. Nomaan Qureshi Mandyam Brunda Samarth Brahmbhatt K. Madhav Krishna Pranjali Pathre, Gunjan Gupta. Imagine2servo: Intelligent visual servoing with diffusion-driven goal generation for robotic tasks. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[12] SRI International. Artificial Intelligence Center and G.J. Agin. *Real time control of a robot with a mobile camera*. Technical note. SRI International, 1979.

[13] Arthur C. Sanderson and Lee E. Weiss. *Adaptive Visual Servo Control of Robots*, pages 107–116. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[14] Francois Chaumette and Seth Hutchinson. Visual servo control. i. basic approaches. *IEEE Robotics Automation Magazine*, 13(4):82–90, 2006.

[15] Frank Hoffmann, Thomas Nierobisch, Torsten Seyffarth, and Günter Rudolph. Visual servoing with moments of sift features. *2006 IEEE International Conference on Systems, Man and Cybernetics*, 5:4262–4267, 2006.

[16] R. Basri, E. Rivlin, and I. Shimshoni. Visual homing: surfing on the epipoles. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 863–869, 1998.

[17] Ezio Malis, Graziano Chesi, and Roberto Cipolla. 2-1/2d visual servoing with respect to planar contours having complex and unknown shapes. *I. J. Robotic Res.*, 22:841–854, 10 2003.

[18] E. Malis, G. Chesi, and R. Cipolla. 212d visual servoing with respect to planar contours having complex and unknown shapes. *The International Journal of Robotics Research*, 22(10-11):841–853, 2003.

[19] Christophe Collewet and Eric Marchand. Photometric visual servoing. *IEEE Transactions on Robotics*, 27(4):828–834, Aug 2011.

[20] Cunjun Yu, Zhongang Cai, Hung Pham, and Quang-Cuong Pham. Siamese convolutional neural network for sub-millimeter-accurate camera pose estimation and visual servoing, 2019.

[21] Quentin Bateux, Eric Marchand, Jürgen Leitner, François Chaumette, and Peter Corke. Training deep neural networks for visual servoing. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3307–3314, 2018.

[22] Aseem Saxena, Harit Pandya, Gourav Kumar, Ayush Gaud, and K. Madhava Krishna. Exploring convolutional networks for end-to-end visual servoing, 2017.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[24] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Deep image homography estimation, 2016.

[25] Y V S Harish, Harit Pandya, Ayush Gaud, Shreya Terupally, Sai Shankar, and K. Madhava Krishna. Dfvs: Deep flow guided scene agnostic image based visual servoing, 2020.

[26] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation, 2018.

[27] Gregory J. Stein, Christopher Bradley, and Nicholas Roy. Learning over subgoals for efficient navigation of structured, unknown environments. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 213–222. PMLR, 29–31 Oct 2018.

[28] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion, 2017.

[29] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning, 2016.

[30] M. Sauvee, P. Poignet, E. Dombre, and E. Courtial. Image based visual servoing through nonlinear model predictive control. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 1776–1781, 2006.

[31] Ian Lenz, Ross Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. 07 2015.

[32] Katharina Bieker, Sebastian Peitz, Steven L. Brunton, J. Nathan Kutz, and Michael Dellnitz. Deep model predictive flow control with limited sensor data and online learning. *Theoretical and Computational Fluid Dynamics*, 34(4):577–591, March 2020.

[33] Pushkal Katara, Y V S Harish, Harit Pandya, Abhinav Gupta, Aadil Mehdi Sanchawala, Gourav Kumar, Brojeshwar Bhowmick, and Madhava Krishna K. Deepmpcvs: Deep model predictive control for visual servoing, 2021.

[34] Hanbo Zhang, Jian Tang, Shiguang Sun, and Xuguang Lan. Robotic grasping from classical to modern: A survey. *arXiv preprint arXiv:2202.03631*, 2022.

[35] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps, 2014.

[36] Sylvain Vandernotte, Abdelhamid Chriette, Philippe Martinet, and Adolfo Suarez Roos. Dynamic sensor-based control. pages 1–6, 11 2016.

[37] Alexander Antonio Oliva, Erwin Aertbeliën, Joris De Schutter, Paolo Robuffo Giordano, and François Chaumette. Towards dynamic visual servoing for interaction control and moving targets. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 150–156, 2022.

[38] Eduardo Godinho Ribeiro, Raul de Queiroz Mendes, and Valdir Grassi. Real-time deep learning approach to visual servo control and grasp detection for autonomous robotic manipulation. *Robotics and Autonomous Systems*, 139:103757, May 2021.

[39] Pablo J. Alhama Blanco, Fares J. Abu-Dakka, and Mohamed Abderrahim. Practical use of robot manipulators as intelligent manufacturing systems. *Sensors*, 18(9), 2018.

[40] Nazib Sobhan and Abu Salman Shaikat. Implementation of pick place robotic arm for warehouse products management. In *2021 IEEE 7th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, pages 156–161, 2021.

[41] Michał Olinski, Paweł Dudziński, and Marco Ceccarelli. Design of a manipulator for agriculture. In Masafumi Okada, editor, *Advances in Mechanism and Machine Science*, pages 651–662, Cham, 2023. Springer Nature Switzerland.

[42] Sayanti Pal, A A Bazil Raj, and Shraddha R Shinde. Performance study of different robotic manipulator systems designed for space debris management. In *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pages 148–153, 2020.

[43] Jingyi Liu, Pietro Balatti, Kirsty Ellis, Denis Hadjivelichkov, Danail Stoyanov, Arash Ajoudani, and Dimitrios Kanoulas. Garbage collection and sorting with a mobile manipulator using deep learning and whole-body control. In *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pages 408–414, 2021.

[44] Sungmin Seung, Pengxin Liu, Sukho Park, Jong Oh Park, and Seong Young Ko. Single-port robotic manipulator system for brain tumor removal surgery: Siromans. *Mechatronics*, 26:16–28, March 2015. Publisher Copyright: © 2015 Elsevier Ltd.

[45] Serdar Kucuk and Z. Bingul. *Robot Kinematics: Forward and Inverse Kinematics*. 12 2006.

[46] Serkan dereli. Iw-pso approach to the inverse kinematics problem solution of a 7-dof serial robot manipulator. *International Journal of Natural and Engineering Sciences*, 36:75–85, 04 2018.

[47] Carlos Faria, Flora Ferreira, Wolfram Erlhagen, Sérgio Monteiro, and Estela Bicho. Position-based kinematics for 7-dof serial manipulators with global configuration control, joint limit and singularity avoidance. *Mechanism and Machine Theory*, 121:317–334, 2018.

[48] H. Seraji. Configuration control of redundant manipulators: theory and implementation. *IEEE Transactions on Robotics and Automation*, 5(4):472–490, 1989.

[49] Pyung Chang. A closed-form solution for inverse kinematics of robot manipulators with redundancy. *IEEE Journal on Robotics and Automation*, 3(5):393–403, October 1987.

[50] Shuxin Xie, Lining Sun, Zhenhua Wang, and Guodong Chen. A speedup method for solving the inverse kinematics problem of robotic manipulators. *International Journal of Advanced Robotic Systems*, 19(3):17298806221104602, 2022.

[51] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edition, 1989.

[52] Fabrizio Caccavale, Ciro Natale, Bruno Siciliano, and Villani Luigi. Resolved-acceleration control of robot manipulators: A critical review with experiments. *Robotica*, 16:565–573, 09 1998.

[53] Minghe Jin, Qiang Liu, Bin Wang, and Hong Liu. An efficient and accurate inverse kinematics for 7-dof redundant manipulators based on a hybrid of analytical and numerical method. *IEEE Access*, 8:16316–16330, 2020.

[54] Xiaohua Shi, Yu Guo, Xuechan Chen, Ziming Chen, and Zhiwei Yang. Kinematics and singularity analysis of a 7-dof redundant manipulator. *Sensors*, 21(21), 2021.

[55] Winfried Lötzsch. Using deep reinforcement learning for the continuous control of robotic arms. *CoRR*, abs/1810.06746, 2018.

[56] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2022.

[57] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. Benchmarking in manipulation research: Using the yale-CMU-berkeley object and model set. *IEEE Robotics &amp Automation Magazine*, 22(3):36–52, sep 2015.

[58] Brandon Amos and Denis Yarats. The differentiable cross-entropy method, 2020.

[59] xarm robot: Ufactory. `https://www.ufactory.cc`. Accessed: 06-04-2024.

[60] Intel® realsense™ depth and tracking cameras. `https://www.intelrealsense.com/depth-camera-d455`. Accessed: 06-04-2024.

[61] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. Instructpix2pix: Learning to follow image editing instructions, 2023.

[62] Zachary Teed and Jia Deng. RAFT: recurrent all-pairs field transforms for optical flow. *CoRR*, abs/2003.12039, 2020.