Cloud-Edge Hybrid Robotic Systems for Physical Human Robot Interactions

Nan Tian

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-142
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-142.html

August 11, 2020

Acknowledgement

I would like to especially thank my PhD Advisor, Professor Somayeh Sojoudi, for her patience, diligence, and optimism when guiding me throughout my PhD; I would also like to thank Professor Ken Goldberg, a committee member and a long time collaborator, who started me on the work of Cloud Robotics and has provided valuable critiques on manuscripts and presentations; and Professor Javad Lavaei, the other committee member, for his continuous support. I would also like to further express my gratitude to Jeff Mahler and Ajay Tanwani for their help and guidance during this collaboration of Cloud Robotic project.

Cloud-Edge Hybrid Robotic Systems for Physical Human Robot Interactions

by

Nan Tian

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Assistant Professor Somayeh Sojoudi, Chair
Professor Javad Lavaeiyanesi
Professor Kenneth Goldberg

Summer 2020

Cloud-Edge Hybrid Robotic Systems for Physical Human Robot Interactions

Abstract

Cloud-Edge Hybrid Robotic Systems for Physical Human Robot Interactions

by

Nan Tian

Doctor of Philosophy in Computer Science

University of California, Berkeley

Assistant Professor Somayeh Sojoudi, Chair

Cloud Robotics is a new paradigm where distributed robots are connected to cloud services via networks to access "unlimited" computation power. Combined with advanced network technology, such as 5G and Wi-Fi 6, it can support service robots operating under unstructured, human rich environments on a global scale. Cloud Robotics has scalable servers that host artificial intelligence, robotic vision, crowd-sourcing, and web-based human computer interface (HCI). These modular Cloud Robotic infrastructures enable control and monitoring of distributed service robots that require sophisticated physical human robot interactions (pHRIs) and human guided tele-operations.

Cloud Robotics is also capable of scale up and down robotic service deployments based on rapid changes in user demands. A similar feature in Cloud-based video conferencing services has shown great value in scaling up and down based on user demands during the on-going Covid-19 pandemic. The ability to match user demands will be an important advantage of using Cloud Robotics to keep the operational cost down for service robots applications, where mixed Cloud Robotic modules can be selected for different environments on demand.

Besides above advantages, Cloud Robotic systems pay the additional price of network communication. There are three major network communication costs that hinder effective deployment of cloud robotics: (1) network bandwidth, (2) privacy and security, (3) network latency and variability. With the emerging high speed 5G and Wi-Fi 6 technology, the cost of network speed and bandwidth are dropping significantly, hence the value of Cloud Robotic services will eventually triumph the cost of network communication. However, if we want to use Cloud Robotic services to control dynamic, compliant, service robots with feedbacks, unpredictable variable delays caused by network routine protocols over long physical distances presents a major obstacle.

In this thesis, we propose a Cloud-Edge hybrid robotic system to enable dynamic, compliant, feedback controls for physical human robot interactions (pHRIs). Specifically, we

built a framework to (1) move centralized high-level controllers and computational intensive perception services to the Cloud; (2) deploy low latency, agile, Edge Robotic controller to handle dynamic and compliant motions; (3) implement a hybrid, two-level feedback controller leveraging both the Cloud and the Edge; (4) use robotic-learning algorithms to perform motion segmentation and synthesis to mitigate network latencies within the Cloud-Edge perception feedback loop. We demonstrate the robustness of the above framework using different robots, including a dual arm robot (Yumi) from ABB, a dynamic self-balancing robot (Igor) and a compliant 5 degree-of-freedom (DoF) robot arm both from Hebi Robotics, and a humanoid robot (Pepper) from Softbank Robotics. A copy of the dissertation talk including video demonstrations can be found here: https://drive.google.com/drive/folders/1rh8gCydsXCpGJCI6n31mwgTdsJdjJfn-?usp=sharing

To my wife, Zhaoyu (Amy) Meng, the love of my life

To my parents, Ping Tao and Zeming Tian, for their kindness and courage

To all my teachers, for their wisdoms and guidances

And to my grandparents, for fond memories

Contents

List of Figures

List of Tables

Acknowledgments

I would like to especially thank my PhD Advisor, Professor Somayeh Sojoudi, for her patience, diligence, and optimism when guiding me throughout my PhD; I would also like to thank Professor Ken Goldberg, a committee member and a long time collaborator, who started me on the work of Cloud Robotics and has provided valuable critiques on manuscripts and presentations; and Professor Javad Lavaei, the other committee member, for his continuous support. I would also like to further express my gratitude to Jeff Mahler and Ajay Tanwani for their help and guidance during this collaboration of Cloud Robotic project.

Over the years, I have been heavily influenced and supported by many brilliant professors at UC Berkeley and from other academic institutes. Although it is hard to address all, I will acknowledge those who directly contribute to the finish of this thesis. I would like to thank Professor Ruzena Bajcsy who first got me interested in robotic research through her Active Perception class, Professor Anca Dragan for her course on human robot interactions, Professor Clair Tomlin for interesting discussions about stabilities in dynamic arm control, Professor Ron Fearing for his valuable inputs on visual servoing and feedback systems, Professor Mark Muller on fast perceptions and dynamic control, Professor Jose Gonzales for discussions about low latency synchronous and asynchronous communication protocols, Professor Jack Gallant for interesting discussion about robotics and neuroscience, Professor John Canny and Professor Travor Darrel on teaching me deep learnings, Professor Howie Choset from CMU and Hebi Robotic for discussions about dynamic compliant robots for human assistances, Professor Jitendra Malik who got me interested in visual perception and machine learning in the first place through his famous Computer Vision and Machine Learning courses, and Professor Philip (Flip) Sabes from neural science at UCSF who first introduced me to the world of computational neuroscience via visuo-motor behaviors and brain machine interface research at UCSF before UC Berkeley.

Further, this work would not have been possible without inputs and joint efforts from peer graduate and under-graduate students. Matthew Matl, Chris Corea, Aashna Garg (Stanford), Michael Yu, and Billy Zhu (CMU) for collaboration and joint efforts on the Cloud Robotic as a service project; Ron Berenstein for the idea of visual servoing; Jim Ren, Sequoia Beckman and Benjamin Kuo (CMU) for help on the humanoid semaphore project; Sanjay Kristnan, Mike Laskey, Daniel Seita for helps on editing various manuscripts; Hao Su (Stanford, now Professor at UCSD) for discussion about 3D robotic vision; Elizabeth Glista for discussions on dynamic controls; Evan Shellhamer and Forest Iandola for getting me involved in the Caffe auto-tuning project during the early years of Caffe and my PhD; and lastly but not least Alex Rusiano, Yi (Will) Wu, Duo Zhang, Heng Chang for being great friends.

I would also like to thank external industrial supports from Matthew Tesch, Bob Raida, David Rollinson, Curtis Layton from Hebi Robotics; Qian Li, Mas Ma, Charles Castello, Rockie Wang, Robert Zhang, and Bill Huang from Cloudminds Inc. Finally, I would like to acknowledge fundings that supported this work: Office of Naval Research pHRI, NSF EPCN, NSF ECDI Secure Fog Robotics.

Chapter 1

Introduction

The world has longed for service robots. Artists, writers, and even musicians from different cultures have fantasized the idea of 'service robot'–robots that assist human–for a long time. For example, in Jacques Offenbach's celebrated opera, *the Tales of Hoffmann*, the master falls in love with his creation–an animatronical doll 'Olympia'; a 'robot venting machine' that was capable of interacting with human appeared in Jules Verne's novel, *the Floating Island*; 'mechanical horses and cattles', described in *Romance of the Three Kingdoms*, help Zhuge Liang's army carry supplies across mountains; and Isaac Asimov's famous 'Three Laws of Robotics' from *I, Robot* inspired us to truly think about service robots and human society. There are also many fictional characters in movies who are service robots–'Data' in *Star Track*, C-3PO and R2-D2 in *Star Wars*, and 'the Terminator' in *Terminator 2: Judgment Day* who defended Sarah Conner.

Now, the world needs 'non-fictional' physical service robots, more than ever. With the on going Shelter-in-Place order in the fight against the Covid-19 global pandemic, we are restrained to study, work, and live at home. We can only communicate with non-family members through tele-conferencing. We can only interact with others in respect to six-feet 'social distancing' guidelines. UC Berkeley even hosted their graduation, *Blockeley* virtually online on Mine Craft.

Service robots are helping us under these unusual circumstances. They help medical professionals monitor and interact with patients in isolation [117], especially elderlies in senior homes who are at high risks of infections [70]. Cleaning robots help hospitals and nursing homes to clean automatically without human contact [126]. Autonomous vehicles help deliver medicine to patients within hospitals [24] or at home [49]. Automatic delivery drones provides contact-less supply drop-offs to people who lives in remote areas [125]. Humanoid social robots even help student in Japan to attend graduation ceremonies physically 'in-person' [107].

1.1 Cloud Robotics for Service Robots

Service robots are different from their counterparts–industrial robots. Industrial robots operate under structured, well controlled environments. They perform repeated, quasi-static tasks on an assembly line that has constant, predictable throughputs. Service robots, however, works in unstructured, human rich environments. They need to perform diverse, dynamic, human compliant, human safe tasks that require constant visual, force, and tactile feedbacks. Further, like video conferencing and other Internet service systems, service robotic systems have variable throughputs based on customer demands. Therefore, they need to be able to scale up and down quickly.

This is where Cloud Robotics becomes useful. Cloud Robotics, introduced by Jeff Kuffner *et al* [75, 66, 65], is a new paradigm based on traditional 'networked robotics'[39]. It uses cloud computing to provide intelligent services to robots with limited computational power. The Cloud, with virtually 'limitless' computation power, connects to distributed robots via Internet, enabling them to perform robotic tasks that they cannot perform before. A recent variant of Cloud Robotics is Fog Robotics [45, 123, 74, 115], where we bring Cloud computing closer to the Edge robotic controllers for more flexible resource placement and allocation.

Both Cloud Robotics and Fog Robotics can be modular [65, 124]. Their modularity makes them easy to maintain, deploy, and propagate so that they can be scaled up and down based on user demand. The Cloud can share data from sensor inputs across different robots. It can share perception streams from the robot to remote tele-operators as well. These streams of shared data over the Cloud can help robots to coordinate and learn from each other. Further, the Cloud can serve as mediator to facilitate data collections from crowd-sourced human demonstrations so that the robots can learn from human behaviors on a global scale [79]. Finally, the Internet connection between the Cloud and the Robot now supports consumer level WiFi or cellular networks [18] rather than traditional wireless RF or satellites connections. These high speed high bandwidth WiFi 6 and 5G networks have become more accessible in recent years for both indoor and outdoor coverages.

1.2 Values of the Cloud Services Triumph Communication Costs

Historically, one major drawback when apply the precursors of Cloud Robotics,such as telerobots [38, 41, 114] and Networked Robotics [91], was the high cost in wireless communication. RF and satellite communications had wide coverage, but they had high delays and limited bandwidth; whereas first generation WiFi had heterogeneous indoor coverage, because their communication protocol had trouble providing multiple devices with balanced bandwidths [62]. Roboticists using Networked Robotics had to struggle with trade-offs between communication costs and limited capabilities in centralized computing resources.

The cost-benefit of Cloud Robotic has changes over time. In the last decade, since the introduction of Cloud Robotics in 2010, high-speed high-bandwidth Internet with CDMA

technology and 4G has become mainstream thanks to popular demands in smart phone usage. The communication costs will continue to drop when emerging 5G and WiFi 6 enter consumer market starting in the year 2020.

Cloud services has also become extremely powerful, backed by hyper-scale data-centers maintained by vendors such as Amazon Web Services, Microsoft Azure, and Google Cloud. These data-center often hosts millions of server nodes, ranging from simple storage nodes to advanced deep-learning servers with multi-core CPUs and multi-GPU. They are all maintained with container based desktop visualization and are publicly available on the market for an affordable price. The Cloud, with 'limitless' compute power, can host sophisticated recommendation systems, deep-learning based perception systems, and crowd-sourced large scale data. With these powerful services, it has the potential to support distributed service robots to perform tasks that was impossible before. As the value of Cloud services is growing and the cost of the communication is dropping, the potential values of Cloud Robotic services will triumph the costs of communication in the near future.

1.3 Network Delays and Variability

Communication costs can be measured in (1) network bandwidth; (2) network security; (3) network delays and variability. High bandwidth 5G and Wifi 6 will lower the network bandwidth costs. Network security ensures privacy of the robotic service users, and are important for the safe usage of Cloud Robotics for service robots. However, it is a complicated problem that touches many fields, including encryption, privacy, system designs, and network administration protocols. We assume it will be resolved by system designers and network security experts as in this work [74].

Network delays and variability, on the other hand, is interesting. They are bounded by physical distances and optimal network routing protocols. They can only be minimized, but the boundary cannot be surpassed due to physical limits. Therefore, we can only mitigate network delays and variabilities, but we can never eliminate them. Moreover, network delays are often un-predictable under heterogeneous Cloud computing environments when they are used to service distributed robots. How to mitigate such un-predictable network delays in Cloud Robotics, particularly in a closed-loop control with perception feedbacks [38], is a major focus of this thesis.

1.4 Physical Human Robot Interactions (pHRIs)

The goal of human robot interactions (HRIs) is to understand and shape the interactions between people and robots [43]. Service robots require physical human robot interactions (pHRIs) to collaborate with human safely and reliably [40] under human rich, dynamic, unstructured environments. Sophisticated perception feedbacks, including visual and tactile feedbacks, are needed for service robots to operate safely and reliably around human.

During the last decade, with the development of inductrial cobots [85] and back-drivable torque controlled modular servos [51], robots have become safer and more compliant. At the same time, robotic vision [136, 22, 50, 37, 60] and tactile perceptions [84, 144] has also made strikes of progress. With both advances in controls and perceptions, we have the fundamental pieces to build fluent pHRI controllers with Cloud Robotics. The key technology that integrates control with perception is feedback control. Therefore, the primary goal of this thesis is to investigate and explore feedback controls for fluent pHRIs under the framework of our 'hybrid' Cloud-Edge system.

1.5 Cloud-Edge 'Hybrid' Dynamical Systems with Feedbacks

There are two ways to use these perceptual feedbacks for robotic controls: (1) traditional *feedbacked controls* where a real-time control loop uses continuous streams of feedbacks to adjust dynamic robotic controls based on predefined control laws [138]; (2) *behavior-based-robotics*, revolutionized by Brooks, Arkin, and others [5], where modes of behaviors extracted from perception map directly to modes of actions so that the robot can be controlled by discrete 'behavior' commands. The former is normally used to control dynamical systems.

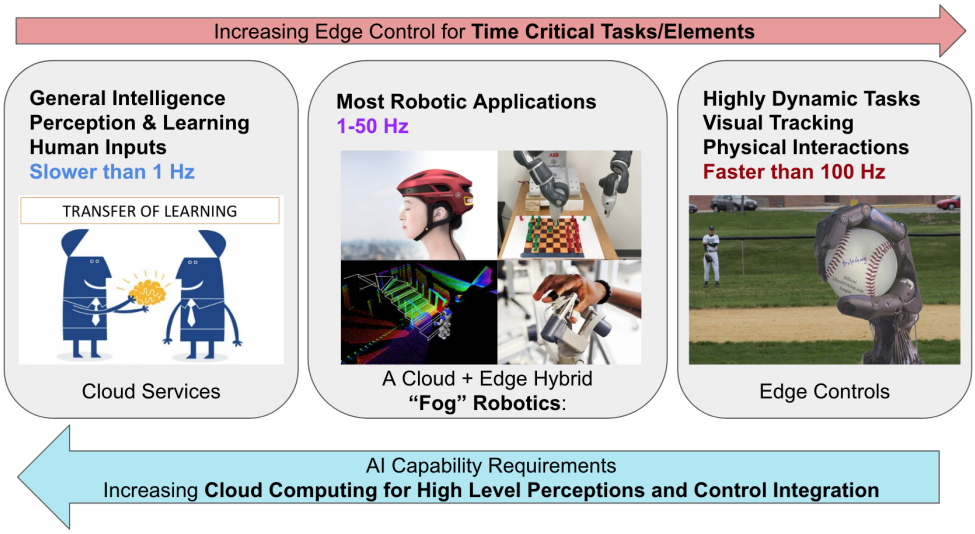

Figure 1.1: Fog Robotics, Intelligence vs. Speed: (Left) the Cloud provide high-level intelligence, such as robotic learning, deep-learning based perception, and cloud based human teleoperation; (Right) the Edge fastest closed-loops controls for highly dynamic robotic tasks, but often lacks high performance computer; (Middle) the Cloud-Edge Hybrid closed-loop control can support majority of the robotic applications.

Since its feedback control loop requires minimum delay, its controller is often local to the robot. However, its perception capabilities are limited due to both the real-time constraint and the limited computational power on the robot. In the latter case of behavior-based-robotics, the control loop is formed based on behaviors. It requires more sophisticated perception to recognize modes of behavior to execute corresponding modes of controls. To process these complex perceptions, more computation is needed, but the control loop can tolerate some delays.

'Hybrid' systems combine the above two kinds of feedback controls so that discrete 'behavior' commands can control a dynamical system [87]. Two feedback loops are formed, a minimum delay real-time inner loop controlled by a high level discrete command loop based on behavior. However, computational power is still a problem if both loop stays local to the robot. Therefore, we build a 'hybrid' Cloud-Edge robotic system to solve this problem. Robots can offload computation intensive perception pipelines, such as deep learning based visual perception module, to the Cloud, while a local low-latency feedback loop is used to execute dynamic motions on the robot. Finally, the Cloud based perception command the Edge dynamic robotic controllers with high level commands to form a second, more intelligent, control loop. Figure 1.2 illustrates two loops formed by behavior feedbacks and real-time dynamic feedbacks in our Cloud-Edge 'Hybrid' system.

pHRIs has into two major areas–teleoperations and proximate interactions. Collaborative, human compliant, and intelligent pHRI controllers that can facilitate fluent interactions are desirable for both teleoperations and proximate physical interactions. Our 'hybrid' Cloud-Edge Robotic system can support fluent pHRI controls for both apllicationsas. Further, as the hybrid system controls both discrete events and continuous real-time motions, its feedback control loop satisfies nearly all service robot application requirements, ranging from quasi-static to highly dynamic (Figure 1.1). We focus mainly on teleoperations throughout in Chapters 2-5, as the Cloud Robotic framework was built to augment human teleoperators under assisted teleoperation settings. We also investigate proximate pHRIs in Chapter 3 and Chapter 4.Chapter 3-5 investigate various forms of such cloud-Edge 'hybrid' robotic systems based on different applications.

1.6 Shared Autonomy with Robotic Learning

The notion of *level of autonomy*, or LOA, is important for 'hybrid' dynamic pHRI systems [43]. In the case of teleoperation using Cloud Robotics, LOAs range from direct teleoperation, to mediated teleoperation, to supervisory control, then to peer-to-peer control, starting from most teleoperator controls with minimum autonomy to less controls with the most autonomy [113]. Different types of shared autonomy defines the coordination between the Cloud and the Edge. They are used to achieve different LOAs required by various robotic applications. How to distribute and use shared autonomy across Cloud services and Edge robotic controllers is a crucial design choice for roboticists to program real-life, pHRI applications. To accommodate most, if not all, forms of shared autonomy, we design a Cloud-Edge Robotic system that

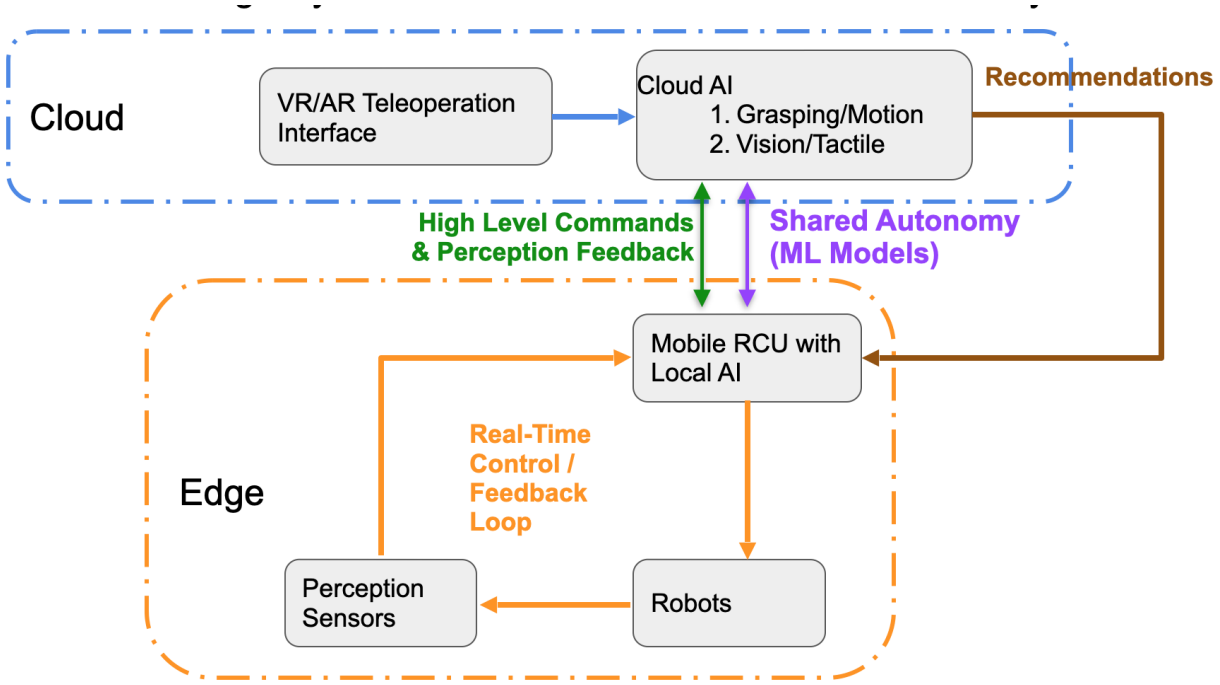

Figure 1.2: Final Cloud-Edge 'Hybrid' System Flow Chart

contains power of Cloud services, responsive Edge robotic controllers, simple programming interfaces and reliable network communication layer that connects the Cloud and the Edge, and most importantly, a 'hybrid' robotic architecture to support various forms of shared autonomy for pHRI systems.

Recent development of robotic learning has revolutionized the field of robotic controls. It allows robots to learn skills or adapt to environments through machine learning. It has improved the performance in the field of object recognition and tracking, dynamic controls, locomotion, graspings, and hand manipulations. Robotic learning can also be used to build advanced shared autonomy systems under our 'hybrid' Cloud-Edge framework. Moreover, such system can help mitigate unpredictable network latencies from Cloud services to the Edge controllers. In Chapter 5, we demonstrate how to use robotic learning to build a shared autonomous hybrid system under the Cloud-Edge framework to mitigate network latencies via motion segmentation and synthesis.

1.7 Thesis Goals and Contributions

The goal of this thesis is to answer below questions:

Can we build a fluent pHRI system using Cloud Robotics? If yes, how?

Hence, we developped a Cloud-Edge 'Hybrid' Robotic system progressively to show that a fluent pHRI system is possible under such novel Cloud Robotic framework. Examples of robotic tasks implemented using our framework are shown in Figure 1.3. Via the development process, we made below contributions:

- Built a Cloud Robotic as a service framework to host a grasping recommendation system, Dex-Net
- Performed multiple pick-and-place robotic task to play chess with non-standard pieces using grasping recommendation served from Cloud grasp recommendation services
- Built a collective learning module in the Cloud to learn from shared empirical data collected by independently controlled robot arms
- Introduced and built a Cloud-Edge robotic framework to control a dynamic self-balancing robot through asynchronous 'heartbeat' protocol
- Achieved autonomous object pickup from random target positions using dynamic visual servoing while hosting visual perception services in the Cloud
- Introduced a 'hybrid' Cloud-Edge framework that can host both intelligent and agile Cloud Robotic applications for service robots
- Used deep learning based gesture recognition system in the Cloud to control a humanoid robot to imitate 'flag language'–Semaphore with pre-generated motion segments stored and executed on the Edge controller
- Benchmarked network latencies across the globe using the Cloud-Edge framework to quantify effects of network delays and variabilities on Cloud Robotic controls
- Identified unpredicted network latencies to be the primary problem in the 'hybrid' Cloud-Edge framework
- Discovered a way to hide network latencies within robot motion execution under the 'hybrid' Cloud-Edge Robotic framework
- Used Guassain Mixture models, Hidden Semi-Markov Models, and Linear Quadratic Tracker to learn models for motion segmentations and synthesis
- Designed and proposed network mitigation protocols under the Cloud-Edge framework using shared autonomy models based on motion segmentations and synthesis techniques

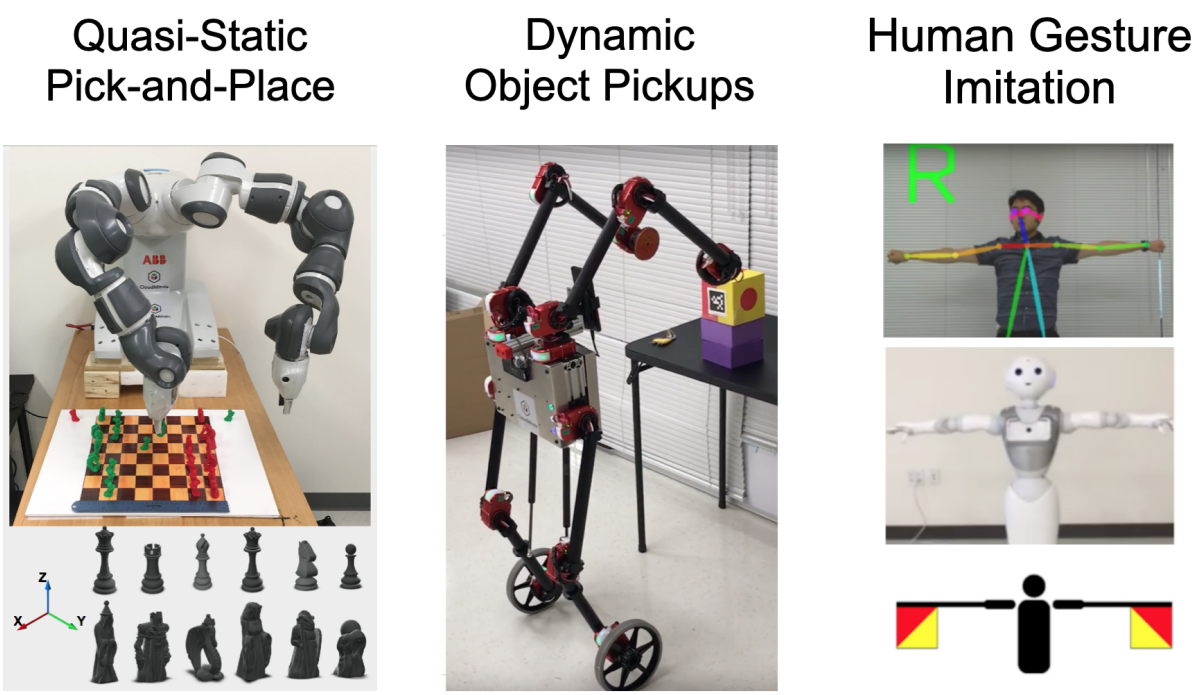

Figure 1.3: Three Example Robotic Application Built with Cloud-Edge 'Hybrid' framework

1.8 Thesis Outline

This thesis contains four chapters of Cloud Robotic works in the progressive order, and a final chapter for discussion.

- Chapter 2 Illustrated the power of Cloud Robotic services using a pick-and-place robotic task–playing chess with non-standard 'wizard' pieces. The system leveraged a Cloud-based grasping recommendation service, Dex-Net, combined with a 'collective learning module' to serve a dual-arm ABB robot YuMi to play non-standard chess reliably. This was a task would have been impossible without Cloud Robotics
- Chapter 3 Demonstrated the first Fog Robotic visual servoing system that could control a dynamic self-balancing robot from the Cloud to pickup objects automatically from a human. It showed that it was possible for the Cloud to control dynamic robots to perform autonomous pHRI tasks, especially when visual detections were done in the Cloud and feedbacked controls were deployed at the Edge robotic controller.
- Chapter 4 Extended the Fog Robotic framework into a 'hybrid' Cloud-Edge Robotic system to support an interactive humanoid robotic tasks that required more sophisti-

cated motion recognition and execution capabilities. The end system could recognize human gesture in the form of semaphore using a deep learning based gesture recognition system in the Cloud. The Cloud then commanded the humanoid robot to mirror the semaphore motion using motions pre-stored at the Edge controller. The work also discovered a latency hiding scheme through network benchmarks across the globe.

- Chapter 5 first identified the unpredicted network latencies are problematic implementing a reliable 'hybrid' Cloud-Edge Robotic system. It then described a way to use robotic learning and shared autonomy to mitigate network latencies. The chapter used teleoperating handwirtten letter as an example to demonstrate how to use motion segmentation and synthesis model to mitigate unpredictable network latencies. It further proposed network mitigation protocols for dynamic controls using the 'hybrid' Cloud-Edge Robotic system.

- Chapter 6 discussed key developments of above works and propose possible future extensions

Chapter 2

Cloud Robotics as a Service

2.1 Introduction

"Cloud Robotics and Automation" describes robots and automation systems that share data, re-use code, and perform necessary but expensive computation on remote cloud servers. It builds on emerging research in Cloud Computing, Deep Learning, Big Data, and government/industry initiatives such as the "Internet of Things", "Industry 4.0", and "Made in China 2025".

In earlier work [65], we proposed the concept of Robotics and Automation as a Service (RAaaS), the robotics-equivalent of Software as a Service (SaaS) – a model in which software components are hosted on central cloud servers and made available to end users and robots over the internet. Under a RAaaS framework, developers of robotics software would be able to access useful services like path planning, object recognition, and robust grasp planning through web-based interfaces. This could reduce development time for robotics applications substantially – instead of the need to install and integrate a variety of software packages, engineers could learn a common web API. In addition, updates and improvements to cloud-based services would immediately be available to all users, easing software maintenance and providing on-going upgrades for end-users. Furthermore, data collected from each robot connected to the RAaaS framework could be used in large-scale learning algorithms to improve robotic performance across the board. Finally, using RAaaS frameworks would allow robots to take advantage of the significant computational and data storage resources of the cloud. However, network latency could hamper the execution of time-sensitive robotic tasks, and network security becomes a much larger concern when physical robotic systems rely on the internet for proper functionality.

This paper describes the architecture of Berkeley RAaaS (BRASS), a prototype RAaaS system, and reports results from several experiments that explore the costs and benefits of using such a system. In this initial prototype, we focus on providing a robust grasp planning service to users. BRASS currently includes the Dexterity Network (Dex-Net 1.0), a robust grasp planning package that has a database of over 10,000 3D object models and

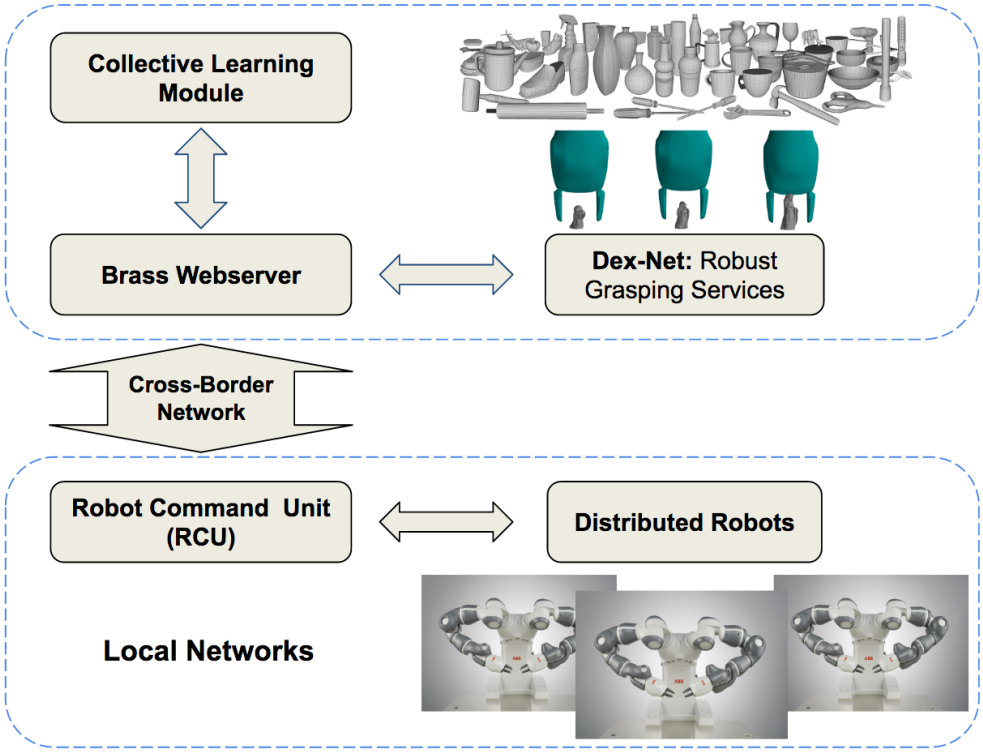

Figure 2.1: Brass Block diagram: (Top) Dex-Net, Brass webserver, and collective learning module on a remote server. (Center) TCP-IP Network such as the Cross-Border Network that enables communication between server and robot systems. (Bottom) Robots with robot command unit (RCU) that locally identifies objects and queries BRASS for grasp configurations.

uses over 1500 cloud computing nodes to pre-compute stochastic robustness properties for hundreds of parallel-jaw grasps per object [88]. Users can connect to BRASS via a standard network or through a proprietary cross-border network provided by Cloudminds Technology, which offers better security features and reduced latency. Once a connection is established, a robot can retrieve hundreds of grasp candidates and their associated robustness metrics from Dex-Net to use as a guide for planning manipulation tasks.

Because robust grasp configurations are pre-computed and stored in the cloud, Dex-Net can empower robots with limited memory and computing power to manipulate complex objects encountered in tasks such as warehouse order fulfillment and home decluttering. Furthermore, users of BRASS do not have to install any software to take advantage of Dex-Net's services. A simple web API exposes all necessary Dex-Net functionality to BRASS users, and updates to Dex-Net's codebase are immediately and transparently available to all usersds. BRASS also includes a collective learning system which stores success rates for

grasps executed by its clients, allowing multiple robots to share their experiences with the goal of collectively learning better grasping configurations over time.

To evaluate the costs and benefits of using Dex-Net through BRASS, we performed three physical experiments with an ABB YuMi bilateral human-safe robot and a set of six asymmetric, non-standard chess pieces downloaded and 3D printed from Thingiverse [108] (see Figure 2.2, bottom). In all experiments, the YuMi retrieves Dex-Net grasp recommendations for each piece through Brass, filters them, and then executes grasps to move each piece. Our initial results suggest that accessing Dex-Net grasp recommendations via Brass and using Brass's collective learning system can increase manipulation success rates with a minor trade-off in network latency.

2.2 Related Work

The concept of cloud robotics and automation can be traced back at least two decades to the advent of "Networked Robotics" [39]. In 1997, Inaba et al. described the advantages of using remote computing for robot control [56], and in 2001 the IEEE Robotics and Automation Society established the Technical Committee on Networked Robotics [54]. Work continued throughout the decade, and in 2009 the RoboEarth project envisioned the construction of "a giant network and database repository where robots can share information and learn from each other about their behaviour and environment" [141, 134]. This project developed cloud computing resources for generating 3D models of environments, speech recognition, and face recognition [127], and the idea of using the cloud for computation and data aggregation in robotics continued to pick up traction. In 2010, James Kuffner first used the term "Cloud Robotics" to describe the increasing number of robotics or automation systems that rely on remote data or code for effective operation [75]. Since then, a wide variety of models for connecting robots to the cloud have been developed, implemented, and tested [65]. Recent work also used cloud computing and deep learning learn hand-eye coordination with up to 14 robot arms [82], along with more recent works in "Cloud Robotics" [57] [61] [30] [119] [109], just to cite a few to show the growing interest.

Some, like RoboEarth's Rapyuta system [95], offer secure, optimized platforms in the cloud for offloading robotic computational tasks. These Platform as a Service (PaaS) systems do not offer robotic services explicitly, but instead make it easier for developers to push existing code into the cloud for parallelization. Adoption of PaaS systems is made easier by the ubiquity of ROS, the Robot Operating System [102], which is organized into nodes that communicate via the ROS messaging protocol. However, these remote nodes are not available as shared services for public users.

On the other hand, some systems have adopted a SaaS-like model for robot motion planning. Vick et al. moved a robot motion controller into the cloud and used it for manipulation planning [133], Zieliński et al. created a cloud system for planning tasks for exploratory robots [147]. Bekris et. al explored the tradeoff between path quality and computational efficiency for a cloud-based motion planner [7], and Ichnowski et. al

Figure 2.2: Dual-Arm YuMi Robot Playing Non-Standard Chess: (Top) The YuMi robot replaying the classic 1997 chess game where IBM's Deep Blue defeated world champion Garry Kasparov with asymmetric "wizard" chess pieces (video: https://youtu.be/_LBBX_oxtbA). (Middle) CAD model of standard chess pieces which can be grasped with hard-coded configurations. (Bottom) CAD model renderings of our non-standard asymmetric chess pieces. The YuMi requires Dex-Net grasp recommendations to manipulate these pieces. The y-axis for each piece points front-to-back, while the x-axis points right-to-left.

split the computation of robot motion plans between the robot's embedded computer and a cloud-based compute service [57]. The SaaS model has also been extended to robotic mappping [6], grasping [14] [27], and cooperative and collective robot learning [119] [44], with distributed partially observed sensor data [42].

Brass builds on ideas from these SaaS-style frameworks as well as related work on robotic grasping systems. In particular, Brass directly uses Dex-Net 1.0 [88] as a service and draws inspiration from Ben Kehoe's work on cloud-based robotic grasping, which used a variety of cloud-hosted services like the Google Object Recognition Engine, OpenRAVE, and the Point Cloud Library in an integrated object manipulation pipeline [66, 64]. Additionally, Brass is based on several ideas from Arjun Singh's dissertation on benchmarks for cloud robotics [112].

Our work is also closely related to robust grasp planning. For a review of grasping, see Bicchi and Kumar [11]. Early research on grasp planning focused on maximizing analytic quality metrics such as force closure [97] or the Ferrari-Canny metric (also known as epslion metric) [32]. However, these metrics depend on precise knowledge of geometry, material properties, contact locations, and surface normals, which may not be known due to imprecision in sensing and control. This motivated the development of robustness metrics, which measure the expectation of quality metrics under uncertainty in variables such as object pose and friction [66, 69, 140]. Since evaluating robustness may be computationally expensive, recent research has studied reducing the number of samples required, for example by using Multi-Armed Bandits (MAB) [78]. Recently, Mahler et al. [88] developed Dex-Net 1.0, a cloud-based dataset of over 10,000 3D models annotated with parallel-jaw grasps and robustness metrics, and showed that this dataset could be used to accelerate robust grasp planning with MAB. Another approach is learning a predictive model of grasp robustness from featurizations such as heightmaps [61] or depth images [58]. In this this work we access robust grasps computed by Dex-Net 1.0 over a network and study their success on a physical system.

2.3 System Design

Our experimental setup consists of the six primary components listed below (Figure 2.1):

1. **A local dual-arm ABB YuMi robot.** This robot receives commands and manipulates chess pieces.

2. **A robot command unit (RCU).** The RCU plans chess moves, makes requests to Brass over the internet to retrieve grasp suggestions, and sends motion commands to the robot.

3. **The Cloudminds Cross-Border Network.** This proprietary network is global, secure, and low-latency, and it is used to connect the local RCU to Brass. It can be

exchanged with an ordinary network when security and reliability is less important to the user.

4. **Brass, a server that exposes a universal API to clients.** Internally, Brass starts up instances of the services it offers, performs queries to each of those services when a web request arrives, and coordinates responses to clients.

5. **An instance of Dex-Net 1.0, run within Brass.** Dex-Net is a service for robust grasp planning that samples parallel-jaw grasps on objects and ranks them by analytic robustness metrics. It currently includes over 2 million grasps for 10,000 object models [88].

6. **A centralized collective learning system, run within Brass.** This service sits between Dex-Net and Brass's external API. It tracks success rates for Dex-Net grasps and modifies grasp recommendations over time.

These components are described in more detail below.

2.3.1 Dual-Arm YuMi Robot

In experiments, an ABB dual-arm YuMi was used as the local robot. This robot has a pair of 7-DOF arms, each of which can move at a rate of $1500\,\mathrm{mm\,s^{-1}}$ with sub-millimeter repeatability and can carry a payload of up to 250 g. Each arm is equipped with a parallel-jaw gripper whose jaws are 5 cm in length and can open to 10 cm apart. The YuMi was designed for manipulating small parts in collaboration with humans, which makes it a good candidate for applications with human-robot interaction (HRI) [1] or humans in proximity.

In our experimental setup, we use the YuMi's ethernet service port to stream RAPID commands for controlling each of its arms independently, and we use YuMi's built-in inverse kinematic solver for planning joint angles.

2.3.2 Robot Command Unit (RCU)

The robot command unit, or RCU, is the mid-level software component that creates manipulation plans, queries Brass to retrieve grasp recommendations, and commands the YuMi to execute grasps on individual chess pieces. The RCU's task planner parses a series of chess moves from a PGN-format chess game file and then begins to execute them sequentially. Throughout the game, the RCU keeps track of each piece's position on the chess board.

When a move is processed, the RCU identifies the target piece and then queries Dex-Net via Brass to retrieve candidate grasps for that piece. These grasps are parametrized by a center point in $\mathbb{R}^3$ that lies halfway between the target locations of two gripper jaws and an axis in $\mathcal{S}^2$ that points from one jaw tip to the other. Additionally, each grasp is returned with its probability of force closure under uncertainty in object pose, gripper pose, and friction,

which serves as our primary quality metric for each grasp in our first set of experiments (Figure 2.3).

To reduce the chance of colliding with other pieces, the RCU constrains the YuMi to grasp pieces from directly above so that the gripper jaws are perpendicular to the table. These grasps can be represented as a rotation of the jaws around the z-axis and a translation of the gripper (see Figure 2.3). From this parametrization, the RCU can directly command the YuMi's built-in motion planner to execute the grasp, lift the piece, and place it in its target location.

2.3.3 Cross-Border Network Service

To connect a local RCU to Brass and its services, we take advantage of Cloudminds' proprietary cross-border network. This network was designed to send files rapidly and securely across great distances, and it differs from traditional networks in two primary ways.

First, this network has control over all routers and access points within its range of service In practice, this results in fewer dropped packets, shorter round-trip times, and reduced network latency when compared to public global networks. Reduced latency and lower-variance round-trip times can make it possible to integrate RAaaS frameworks into time-critical robotics applications.

Second, this network requires every user to authenticate before use and immediately drops packets from unauthorized users. This protocol makes the network more secure and hardens it against denial-of-service attacks, which could result in improved availability, better privacy protection, and safer operation for robotic systems that depend on cloud services.

2.3.4 Brass

Brass is a prototype RAaaS webserver that presents a generic API for robotic services over the internet. By presenting a uniform, well-defined API, Brass abstracts away the complexities of its internal services and enables end-users to build code against a stable interface. Internally, Brass starts an instance of each of its services and connects to them using their own built-in communication protocols. When a request arrives, Brass queries these services, aggregates results, and returns them to the client in a well-defined format. In our experimental setup, Brass receives HTTP requests from the local RCU and sends responses (lists of parametrized grasps) as JSON.

In our current prototype, Brass offers Dex-Net and a collective learning system as services for robust grasp planning. Each request to Brass contains an object identifier, and Brass can either return a raw list of Dex-Net's grasp candidates for that object (sorted by one of Dex-Net's robustness metrics) or a set of grasps recommended by the collective learning system. If the user desires, they can report the success or failure of a grasp by sending a follow-up post to Brass, and that information is then fed back into the collective learning system to improve grasp selections generated by that module in the future.

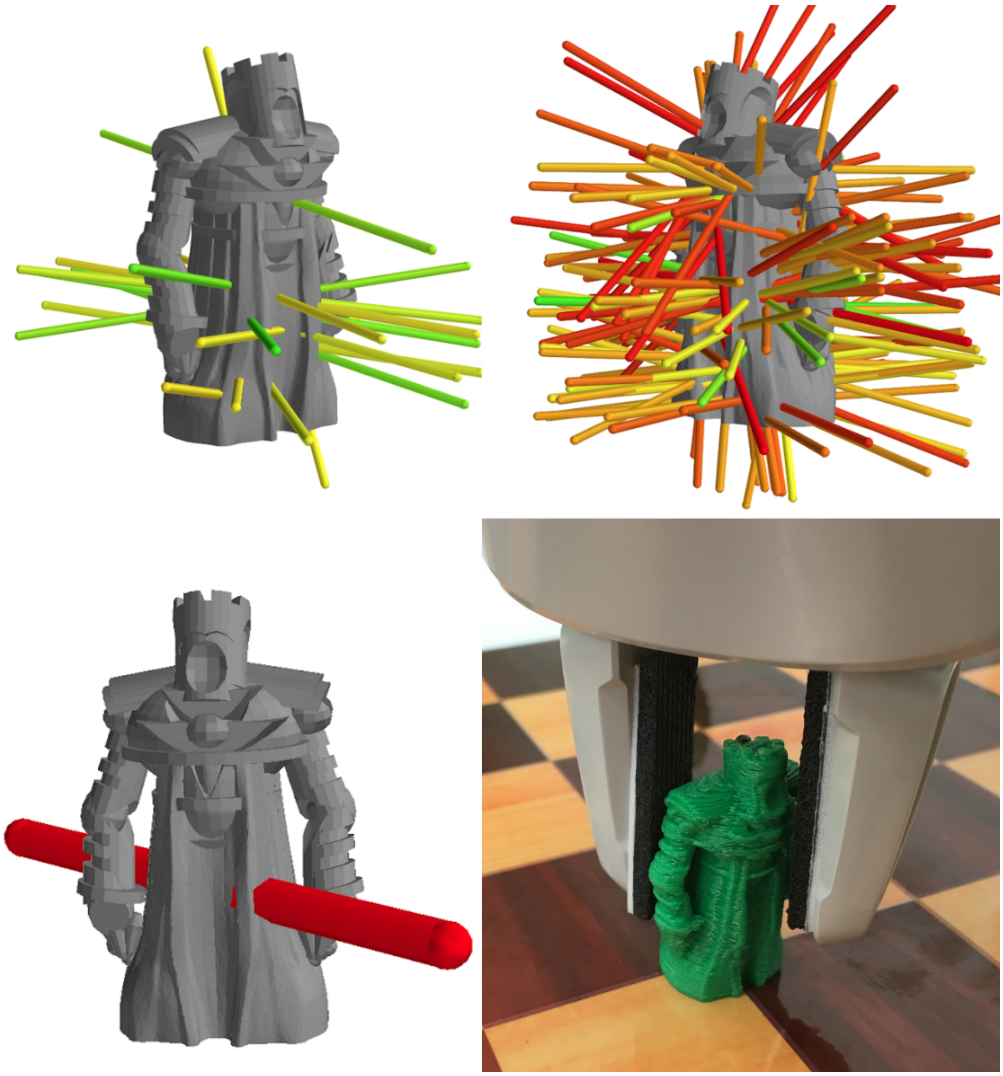

Figure 2.3: Grasping Recommendation Filtering Process: (Top) Parallel-jaw grasp configuration candidates generated by Dex-Net ranked by robustness (probability of force closure). The left image shows the grasp axes for the 15 most robust grasps, while the right image shows the 200 most robust grasps. (Bottom) The left image shows the most robust grasp for the rook piece, while the right shows that grasp being executed by the YuMi robot gripper.

In addition, Brass provides a sample hand-tailored custom heuristic that filters out grasps that are likely to fail in table-top pick-and-place tasks, such as playing chess. In addition to filtering out grasps with a low probability of force closure, this heuristic eliminates grasps whose centers are far from the object's center of mass. This ensures that the gravitational torque on the piece is low when it is lifted. The filter also removes grasps which are too close to the table to reduce the chances of an inadvertent collision and prioritizes grasps whose axis is nearly parallel to the table (due to the vertical grasping constraint imposed by the RCU).

The steps in this filtering process are as follows:

1. Filter out grasps whose probability of force closure is below 30% of the highest probability of force closure over all grasps.

2. Filter out grasps that are within 10 mm of the table to avoid collisions with the YuMi gripper.

3. Filter out grasps whose center point is more than 5 mm from the line that runs through the piece's center of gravity perpendicular to the table.

4. Sort the remaining grasps to prioritize ones whose axes (the vectors in 2.3) are most parallel to the table.

Application of these filtering steps led to the selection of grasps shown in Figure 2.4.

2.3.5 Dexterity Network

Dex-Net 1.0 is a robust cloud-based grasp-planning service [88]. Given a particular object's 3D mesh model, Dex-Net computes all stable poses for the object on a planar worksurface using static analysis [142]. Then, Dex-Net generates thousands of grasp candidates for each registered gripper type using a multi-armed bandit model to leverage prior grasps for increased sampling efficiency. Individual grasps are represented as a center point and a 3D vector that indicates the axis between the jaws. Finally, robustness metrics, such as probability of force closure [139] and expected Ferrari-Canny quality [68] [31] are computed for each grasp under uncertainty in object pose ξ, gripper pose ν, and friction coefficient γ, and these metrics are stored with each grasp for later access.

The grasp sampling and metric calculation processes are computationally expensive, requiring 10-15 minutes per object on a standard desktop, so Dex-Net takes advantage of cluster computing and uses 1500 nodes to pre-compute grasps. Dex-Net currently stores data for over 10,000 unique objects, which amounts to over 2.5 million grasp candidates. Dex-Net can then serve these pre-computed grasps over the internet via Brass in small, kilobyte-sized packages. This means that nearly any system – even ones constrained by both

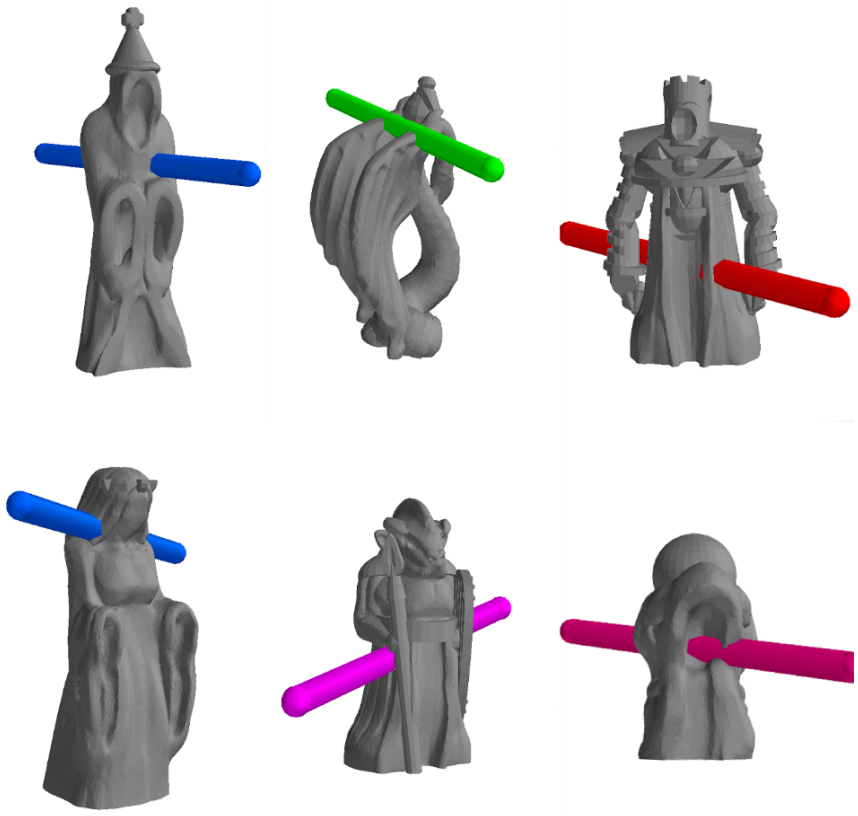

Figure 2.4: Visualization of the Dex-Net grasps selected by Brass's heuristic. When these grasps are executed, the axis shown is projected onto a plane that contains its center point and is parallel to the table.

memory and computational power – can use Dex-Net to plan robust grasps for thousands of objects.

For these experiments, the 3D mesh model for each chess piece was added to Dex-Net and grasps were pre-computed before the experiments were initiated.

2.3.6 Collective Learning System

Dex-Net uses perturbation sampling to estimate stochastic robustness metrics for each grasp. These physical perturbations, such as minor deviations in positions and rotations of the object, are modelled as Gaussian distributions. Prior work has shown that grasps generated with this method perform well in practice [88], and our first major experiment supports these results. However, this stochastic model may not fit every kind of disturbance that could occur in practice. For example, systematic errors introduced by sensor bias or imperfect calibration could violate Dex-Net's assumptions about noise, and variable gripper surface

types could change the performance of individual grasps. Therefore, collecting large amounts of empirical data can still be useful to further understand and improve the performance of grasping systems. However, running large-scale physical experiments is difficult and time consuming, especially if only a small number of robots are available.

RAaaS frameworks such as Brass have a potential solution to this problem – robots connected to the framework can share their individual experiences so that all connected robots can learn from them. This effectively amplifies the amount of empirical data available to each robot, allowing for large-scale collective learning with low per-robot load. In Brass, this idea is implemented in the collective learning module.

The collective learning module is initialized with a set of grasp candidates from Dex-Net. When robots connected to Brass execute one of these grasp candidates, they can send a report back to Brass that indicates whether the grasp succeeded or failed. These reports are saved in the collective learning module, and after enough trials are collected for a particular grasp, the module can start to provide recommendations from its set of candidates, favoring those with the highest empirical rates of success.

In our experiments, we show that collecting empirical grasp data can help to find robust grasps when high-quality metrics are not available. In addition, this module serves as a proof-of-concept for other potential RAaaS modules that could leverage the experiences of many robots to improve performance at scale.

2.4 Experiment and Result

In order to evaluate the costs and benefits of using Dex-Net through Brass, we performed three sets of physical experiments with the ABB YuMi robot and our set of asymmetric chess pieces. Each of these experiments is described in detail in this section.

Table 2.1: Grasp Success Rates (percentage) over 50 Attempts

	Hardcoded x-axis	Hardcoded y-axis	Dex-Net
King	94	100	100
Queen	84	100	98
Rook	96	98	100
Bishop	98	72	98
Knight	100	100	100
Pawn	94	76	100

2.4.1 Individual and Sequential Grasp Success With and Without Dex-Net

Our first experiment evaluated how using Dex-Net grasp recommendations affects grasp reliability when compared against hard-coded grasps that do not take piece geometry into account. This experiment was divided into two scenarios:

1. Dex-Net is not used. Instead, the tip of the gripper is positioned to grasp the center of mass of each piece with the jaws placed parallel to either the x-axis (left to right) or y-axis (front to back, see Figure 2.2).

2. Dex-Net is used via Brass to provide robust grasp candidates. Brass performs grasp filtering using our custom heuristic with robust force closure to provide the RCU with a single target grasp.

For each scenario, we attempted to grasp each piece 50 times and move it from a start square to a target square. This experiment was designed to characterize the probability of correctly manipulating each individual piece under each of the listed scenarios. An attempt was considered successful if the piece was successfully lifted from its start square and placed in the target square without being dropped or knocked over. After each test, the target piece was re-registered in its start square. The results from these experiments are displayed in Table 2.1.

2.4.2 Collective Learning Experiment

As a proof-of-concept for how robots can share empirical data to learn together through a RAaaS system, we performed a test of Brass's collective learning module.

Table 2.2: Success Probabilities (Percentage) of the 5 Grasps based on 30 Physical Trials

Grasp	1	2	3	4	5	Best
King	**100**	100	100	100	100	**100**
Queen	**100**	97	87	67	100	**100**
Rook	**100**	87	100	100	100	**100**
Bishop	57	87	**100**	60	100	**100**
Knight	87	63	**100**	100	27	**100**
Pawn	47	50	**100**	100	100	**100**

In this experiment, we initialized the collective learning module with five grasps from Dex-Net for each chess piece. Ordinarily, Dex-Net would recommend grasps ranked by its robustness metrics, which would produce a top set of grasps that would all work well in practice. However, in order to showcase the improvements that empirical learning could produce if Dex-Net's assumptions were violated, we intentionally selected the five candidate grasps from Dex-Net *without* considering any of Dex-Net's robustness metrics.

Instead, these grasps were selected using steps 2, 3, and 4 from Brass's hand-tuned heuristic, which merely ensures that the selected grasps are near the center of the piece and do not collide with the table. Such a selection results in enough variability in grasp quality among the candidates for the collective learning system to be useful.

Once the collective learning module was initialized with candidate grasps, we connected the left and right arms of the YuMi to Brass as separate robots. Then, we performed fifteen individual attempts per grasp with each arm. After each trial, the arms each forwarded their results back up to Brass's collective learning module. After all candidate grasps had been tried at least fifteen times per arm, the learning module began recommending the most successful grasp for each piece upon future requests and continued to update its empirical database dynamically.

Empirical success rates for each of the five candidate grasps for each piece are shown in Table 2.2. As expected, some of the grasps selected by Brass's imperfect heuristic performed quite poorly. However, by working together and sharing data, the YuMi's two arms were able to converge to grasps with high success rates.

As an exmpale, Figure 2.5 top illustrates the visual demonstration of the 5 grasps selected by Brass's imperfect heuristic for the bishop piece, and table 2.2 row 4 shows the empirical results collected via collective learning and physical experiments. The third-ranked grasp had the highest empirical success rate, and the collective learning system enabled the YuMi's arms to jointly teach it to choose that grasp for future trials. While grasps 2-5 for the bishop (Figure 2.5) all look similar, they had very different success rates. Grasp 2 failed frequently because the bishop's slanted surface slipped out of the grippers when pressure was applied, and grasp 4 failed when the gripper collided with the piece's shoulder above it. Grasps 3 and 5 worked well because the surfaces of contact were nearly parallel and admitted the gripper without collisions.

As a side note, Dex-Net's robustness metrics *did*, in fact, predict these differences in performance, which illustrates the usefulness of having good analytic metrics for filtering grasp candidates. In the absence of such metrics, a seemingly reasonable heuristic produced some undesirable grasp candidates, but we were able to recover by using the collective learning module to help both arms discover good grasping configurations. In practice, users could use both Dex-Net's robustness metrics and Brass's collective learning module to maximize success rates in manipulation tasks.

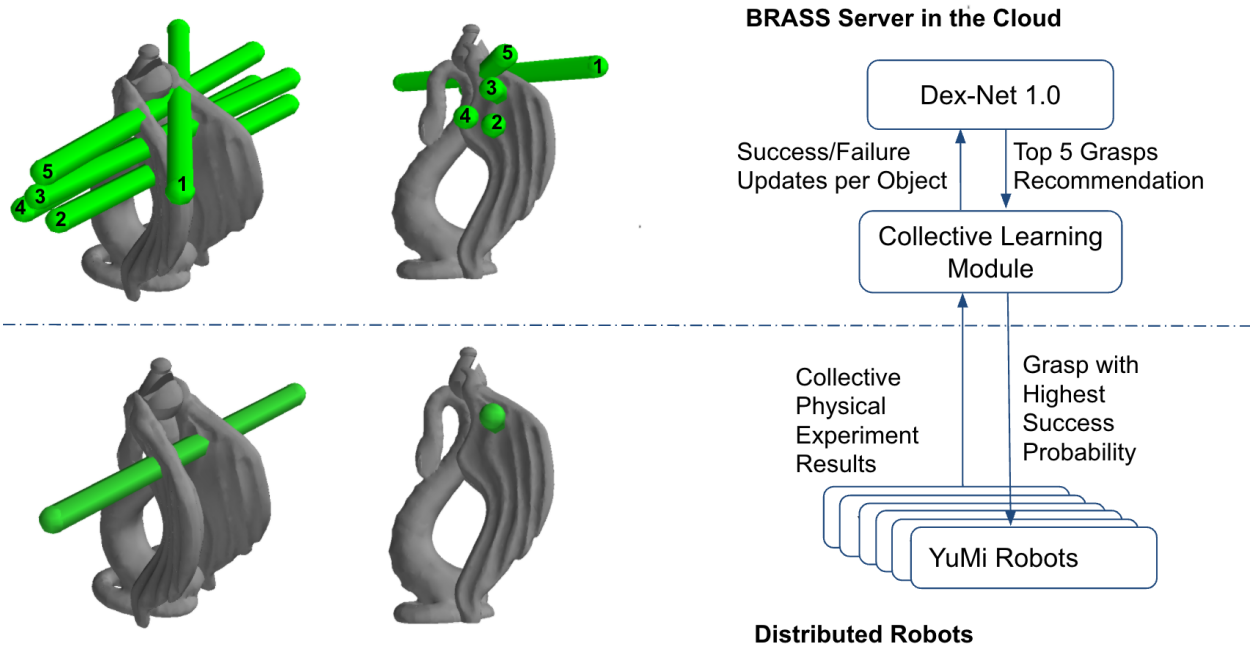

Figure 2.5: Grasps Selection using Collected Learning Module (Left Top) The bishop chess piece with 5 grasps proposed by Brass's heuristic. (Left Bottom) One of the best grasps was grasp 3, with a 100% probability of success as determined by 30 physical experiments with two independently controlled robot arms and the collective learning module in the cloud. (Right) Block Diagram of the collective learning system.

2.4.3 Chess Movement Time vs. Network Latencies

In our final experiment, we benchmarked the network latency and variance for accessing Brass servers in a variety of scenarios. Four primary configurations were considered:

1. Brass, Dex-Net, and the RCU are all deployed on a local machine directly connected to the YuMi.

2. Brass and Dex-Net are deployed on an Amazon EC2 instance in Oregon, while the robot is in San Francisco. An ordinary ISP's network is used for packet transportation.

3. Brass and Dex-Net are deployed on a server in China, and the robot is in San Francisco. An ordinary ISP's network is used for packet transportation.

4. Brass and Dex-Net are deployed on a server in China, and the robot is in San Francisco. Cloudminds' proprietary network is used for packet transportation.

For each scenario, we bench-marked the round-trip time to perform a Brass query for Dex-Net grasp candidates via TCP/IP. Each query contained the name of the target piece, and each response contained a raw list of Dex-Net's pre-computed grasps for that piece in JSON format. Responses were generally around 33 kB in size. One hundred trials were performed per scenario, and the results are shown in Table 2.3.

Table 2.3: Round-trip Time to Receive Grasps from Dex-Net

	Mean (ms)	Variance (ms)
Local	0.11	0.0011
Oregon, EC2	31.50	0.0018
China, normal net	303.10	54.2180
China, cross-border net	197.03	0.3239

2.5 Discussion

In the first experiment, grasp performance for three of the chess pieces was robust across all methods, but grasp performance for the pawn, bishop, and the rook was better with Dex-Net than without. These experiments suggest that utilizing Dex-Net via Brass can provide substantial gains in grasp robustness at the cost of some network latency.

However, this cost must be explored, since network latency is an important consideration for cloud-enabled robots. For time-sensitive on-line and real-time applications, large latencies for critical services can render a system unresponsive or unstable. For example, if an assembly-line robot needed to access Dex-Net grasps for objects moving on a conveyor belt, a 200 ms network delay could significantly impede performance.

Fortunately, the Amazon EC2 configuration demonstrated 32 ms latency with relatively low variance for domestic communication, and Cloudminds' network showed 197 ms latency and low variance communication between San Francisco and China (see Table 2.3). Thus, a reliable 30 Hz control signal from a RAaaS system can be achieved if the server is in the same geographic region as its clients, which is adequate for most on-line and some real-time robotics applications. For servers that are far away, even with the state of the art network technology, only a 5 Hz control signal is available, which likely rules out most real-time applications. However, when the connection is reliable, RAaaS systems at such long distances can still be useful for on-line applications with looser latency requirements. For these types of applications, RAaaS systems can provide nearly global coverage.

In addition to showing the benefits of connecting robots to advanced services such as Dex-Net, we also showed how RAaaS systems can enable the sharing of empirical data for collective learning. In our experiment with Brass's collective learning module, the two YuMi

arms worked together to discover a better grasp than the one suggested by Brass' hand-tuned heuristic for the Rook, Bishop, Knight, and Pawn pieces. This provided a proof-of-concept for how robots could use RAaaS systems to share data in order to empower machine learning algorithms, amplify the amount of empirical data available to individual robots, and reduce the experimental load on each robot while improving performance.

Finally, we will examine ways to enrich Brass's collective learning module in the future. One idea is to use data cleaning techniques [71] [72] to detect systematic robotic failures based on outliers from recorded experimental data.

2.6 Additional Cloud Robotic Service Modules

Since the original Brass paper, we have built more Cloud modules around the non-standard chess playing system to demonstrate the power of modularity using Cloud Robotics. These modules, demonstrated in Figure 2.6, include (1) immersive VR/AR teleoperation module in the Cloud, (2) web-based teleoperation interface that stream videos for the user to play non-standard chess remotely, (3) 3D visual correction system in the Cloud to use visual feedbacks to further improve the reliability of the non-standard chess playing system. Video demonstrations can be found at https://drive.google.com/drive/folders/1rh8gCydsXCpGJCI6n31mwgTdsJdjJfn-?usp=sharing

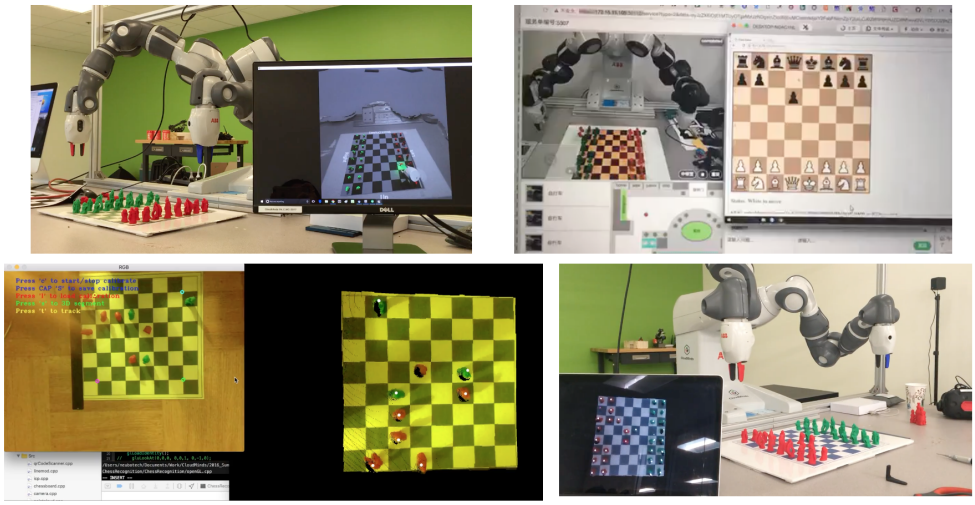

Figure 2.6: Additional Cloud Robotic Service Modules: (top left) Immersive VR/AR teleoperation module; (top right) web-based teleoperation interface with video interface; (bottom) 3D Visual correction module using point clouds and surface matching to track the centroid of the non-standard chess pieces.

Notes

Part of this chapter has appeared in a peer-reviewed conference paper:

Nan Tian, Matthew Matl, Jeffrey Mahler, Yu Xiang Zhou, Samantha Staszak, Christopher Correa, Steven Zheng, Qiang Li, Robert Zhang, Ken Goldberg. *A Cloud Robot System Using the Dexterity Network and Berkeley Robotics and Automation as a Service (BRASS).* ICRA 2017.

Chapter 3

A Fog Robotic System for Dynamic Visual Servoing

3.1 Introduction

Service robots operate semi- or fully autonomously to perform services useful to the well-being of humans and equipments[104]. International Federation of Robotics (IFR) predicts that 32 million service robots are to be deployed between 2018-2022 [103]. Some popular service robot applications include elderly care, house cleaning, cooking, patrol robots, robot receptionists, entertainment, and education. A few famous examples of service robots are Roomba by iRobot, Pepper by Softbank Robotics, "the robotic chef" by Moley Robotics, and Spotmini and Atlas by Boston Dynamics.

Different from industrial robots, service robots need to interact and cooperate with people safely under dynamic unstructured environments. We believe there are two key requirements for service robot operations: (1) accurate, general visual perceptions as feedbacks, and (2) intelligent, dynamic, human compliant robotic controls.

With recent breakthroughs in deep neural networks and robotic learning, robot visual perceptions [36][73][2][23] and intelligent controls [89][81][82][34][80], learning-based intelligent service robot systems can start to operate under unstructured, dynamic, human compliant, general environments. However, these learning-based technologies come with a high computation cost, yet, normal robotic controllers have limited computation power. It is hard to deploy all of them directly on native robot controllers with such constrain.

One solution is to move these computation intensive learning systems into the cloud while keeping the time sensitive dynamic controls on the robot (Fig. 3.1). An example is our previous work on gesture based semaphore mirroring using a humanoid robot [128], where we moved deep-learning-based gesture inferencing into the cloud while executing motion segments locally on the robot.

Current Cloud Robotics systems, however, has high network communication costs, in the form of privacy, security, bandwidth, latency, and variability. Specifically, network laten-

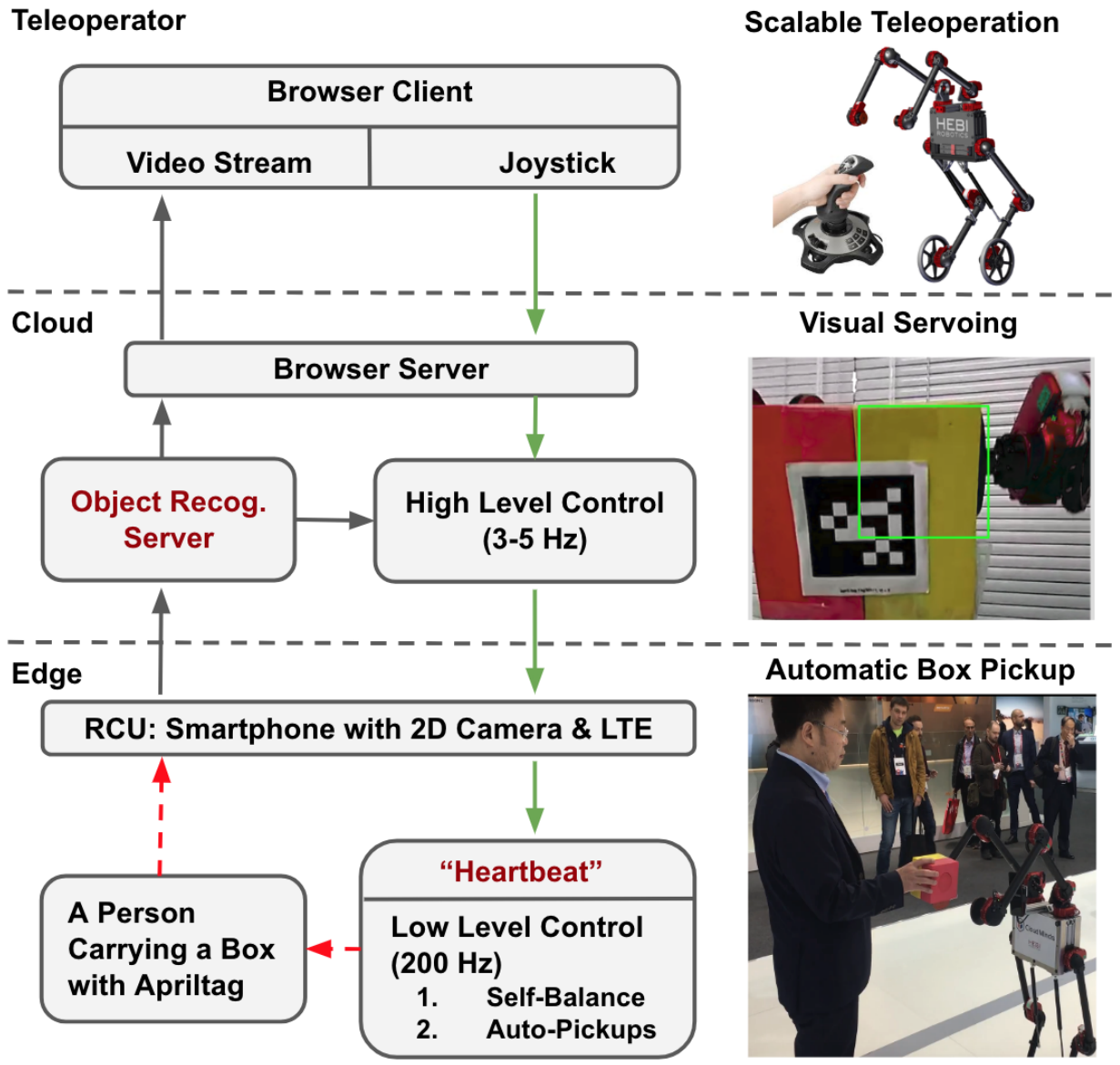

Figure 3.1: The Fog Robotic system for dyanmic visual servoing: (Left) Architecture diagram and information flow–visual perceptions (black arrows), control signals (green arrows), human-robot interactions (red arrows). (Right) Illustrations of three major contributions of this work, teleoperation, visual servoing, and auto box-pickups from a human. They are positioned to their related functional blocks in the Fog Robotic system

cies and variabilities, bounded by speed-of-light and inconsistent network routings, prevent cloud-based robotic controller from controlling dynamic robots directly. Expeccially in human robot interaction applications where visual feedback is required to achieve interactive, human-compliant robot tasks.

To solve this problem, we combine both powerful cloud services and agile edge devices to build an intelligent Cloud-Edge hybrid control system under a Fog Robotic framework. We want to demonstrate that, with this hybrid design, not only a Fog Robotics can expand the robot's general "intelligence" with cloud based AI modules, but it can also effectively control a dynamic robot at the Edge, even when network connections are not perfect.

Therefore, we choose to perform a robotic task that is commonly performed in warehouse logistics, namely box pickups, using a dynamic, dual arm, dual leg, self-balancing robot named Igor, made by HEBI robotics. We build an automatic box pickup module to show: (1) cloud-based assisted teleoperation; (2) visual recognition in the Cloud; (3) dynamic self-balancing navigation controller at the Edge; (4) a closed-loop Cloud-Edge hybrid controller for automatic box pickups using visual feedbacks (Fig. 3.1). This hybrid module is implemented under Human Augmented Robotic Intelligence Platform, or HARI [128], provided by Cloudminds Inc.

Further, we choose to implement this hybrid visual feedback controller with image based visual servoing (IBVS) to eliminate the time-consuming but often necessary robot-camera registration routine. With Fog Robotic IBVS, we make the box pickup system more practical, so that Igor can perform efficient, reliable, automatic box pickups from a human carrier under unstructured environments.

3.2 Contribution

1. **A Cloud-Edge hybrid controller** for reliable teleoperation of a dynamic self-balancing robot.

2. **An autonomous Fog Robotic IBVS module** that assists cloud teleoperators for efficient object pickups.

3. **Automatic box-pickups from a moving human** to demonstrate dynamic human robot interactions (HRI).

3.3 Related Work

Cloud Robotics refer to any robot or automation system that relies on either data or code from a network to support its operation [65]. The term was introduced by James Kuffner in 2010. It was evolved from *Networked Robotics* [75]. Well-known Cloud Robotic Systems includes: RoboEarth's Rapyuta [95], motion planning for services at both cloud [133] and edge [57], Berkeley robotics and automation as a service (Brass) [129], and Dex-Net as a

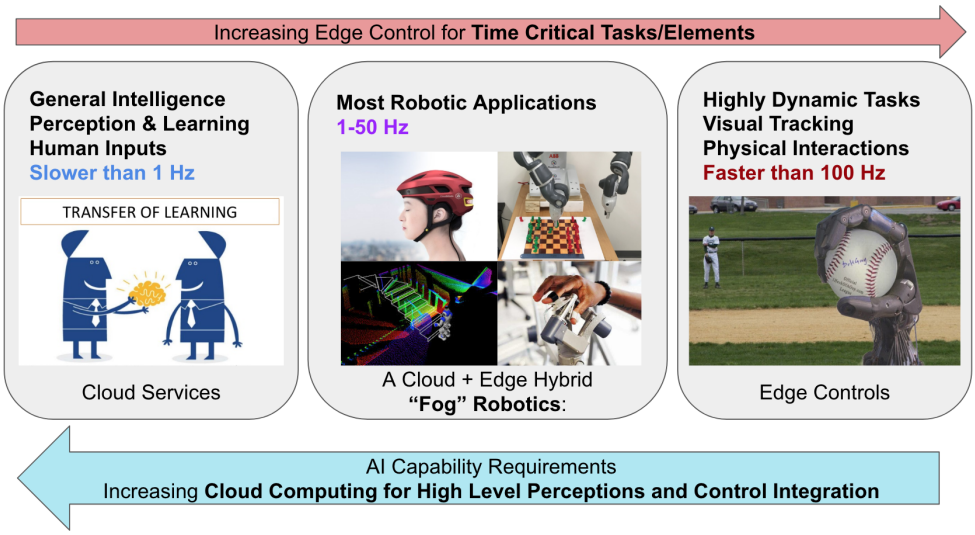

Figure 3.2: Fog Robotics, Intelligence vs. Speed: (Left) the Cloud provide high-level intelligence, such as robotic learning, deep-learning based perception, and cloud based human teleoperation; (Right) the Edge fastest closed-loops controls for highly dynamic robotic tasks, but often lacks high performance computer; (Middle) the Cloud-Edge Hybrid can handle majority of the robotic applications.

Service (DNaaS) [83, 90, 89], just to name a few. However, network costs in the form of privacy, security latency, bandwidth, and reliability present a challenge in Cloud Robotics [124].

Fog Robotics is a form of Cloud Robotics in which cloud computing resources is brought closer to the robot to balance storage, compute and networking resources between the Cloud and the Edge [124]. It is inspired by Fog Computing, originally introduced by Cisco Systems in 2012 [15] [96], and it was recently introduced by Gudi et al [45]. A Fog Robotic system was developed and evaluated in parallel at Berkeley by Tanwani et al for learning surface grasps from nearby environments [124]. There are also other works on Fog Robotics that focus on investigating optimal off-load strategies [63] [26].

Fog Robotics provides a general cloud framework to combine specialized yet complimentary intelligent systems to work together, so that users can build a scalable HRI cloud service. The Cloud can host advanced robotic learning systems, such as grasping and decluttering [88], [89], [124], visual servoing [80], guided policy search [81], [82], visual foresight [34], and domain randomization [131]. It can also host deep-learning based vision systems for object detection [73] [36] [2] and human gesture recognition [23][22] in semi-realtime (5 - 10Hz) to provide visual inputs to the Edge device. Such Cloud-Edge hybrid system, combining both intelligence and speed, can cover a full spectrum of tasks for service robots (Fig. 3.2). One example is our recent work on a cloud-based sempahore imitation system using a hu-

manoid robot [128], but we believe a more dynamic task like control a self-balancing robot can demonstrate the power of this hybrid system even further.

Latency and Variability contributed by network imperfections and cloud computation [124] are the main caveats to control a dynamic robot with Cloud and Fog Robotics. These imperfections distort the controllability of a dynamic system, especially with a real-time close-loop feedback controller. While Cloud Robotics often include some capacity for local/edge processing for low-latency, real-time responses [65], how to design a reliable cloud-edge hybrid, dynamic feedback controller is still an interesting open problem. This problem is similar to previous work on networked control systems (NCSs) [146][143] [137] which assumes either centralized or distributed controllers, but different, because the Cloud and the Edge host a closed-loop controller together in Fog Robotics.

Intelligent Visual Feedback Systems has been used for AI assisted teleoperation to improve teleoperator's efficiency under harsh environments [12], [48]. However, dynamic visual feedback system under unstructured environment is challenging because traditional industrial robotic vision approaches are developed for precision under highly controlled manufacturing environments. Time consuming registration [10] and calibration [132] are often required before performing a robotic task, not practical for many service robotic tasks with HRIs. We choose to implement a Image Based Visual Servoings (IBVS) controller [53], [25], [111] which uses dynamic feedback control policy to minimize relative distance between the robot and the target in 2D image space. There are previous works on Jacobian estimation for robust visual servoing [111] and NCSs based visual servoing systems [143][137]. We differentiate our work by proposing a hybrid Cloud-Edge framework where the time sensitive, dynamic servoing controller is deployed locally on the Edge.

3.4 Self-Balancing Robot, Igor

Igor is a 14 degrees of freedom (DoF), dual-arm, dual-leg, dynamic self-balancing robot, made by HEBI robotics (shown in Fig. 3.3). Each DoF is built with a self-contained, series elastic X-series servo modules. These servos can be controlled with position, velocity, and torque commands simultaneously, and can provide accurate measurements of these three quantities to a central computer at high speed (>1KHz) with minimum latency. These modules also act as Ethernet switch, and rely sensor informations and commands to and from the native on-board Intel Nuc computer on the robot.

The self-balancing is achieved by modeling the system as an inverted pendulum (see Fig. 3.3 bottom). To estimate the robot's center of mass (CoM), the CoM of the two arms and two legs are first measured in real-time using forward kinematics via HEBI's API. The position of the total CoM is then estimated as the average of the CoMs of the four extremities plus the CoM of the control box weighted by the mass distribution:

$$x_{CoM} = \frac{\sum_i m_i x_i}{\sum_i m_i} \qquad i = \text{arms, legs, box} \tag{3.1}$$

Igor also uses accelerometer measurements to estimate the direction of gravity (G) at all times. With CoM of Igor, center of wheels (o_w), and direction of gravity, we can calculate the length and direction of the inverted pendulum:

$$L = x_{CoM} - o_w \tag{3.2}$$

The lean angle (ψ), which is the angle between gravity and the inverted pendulum can then be estimated in real-time:

$$\psi = cos^{-1}\Big(\frac{L \cdot G}{|L||G|}\Big) \tag{3.3}$$

To keep the robot balancing, we assume that the lean angle (ψ) is small so that the system can be linearized:

$$\psi \cong sin(\psi) \tag{3.4}$$

torque ($\mathcal{T}$) in the direction of falling is applied to the wheel with radius (R) and angular velocity (ω) to counteract the effects of gravity on the robot's center of mass:

$$\mathcal{T} = R\omega = v - \dot{\psi}L \tag{3.5}$$

where v is the velocity of the robot's CoM. Furthermore, the derivative of the lean angle ($\dot{\psi}$) can be controlled by a proportional controller with coefficient (K_v), and is related to the velocity of the robot as follows:

$$\dot{\psi} = K_v v \tag{3.6}$$

Real-time measurements of both robot CoM and direction of gravity are important, because the Igor controller needs to compensate for dynamic movements of the four extremities for robust self-balance control. We can also rotate the robot by applying different velocities

to the two wheels. Inertia measurement unit (IMU) readings can then be used to control robot turning angles in real-time.

3.5 An Intelligent Fog Robotic Controller

3.5.1 Edge Controllers

There are two edge controllers in our system. The first is the native Igor robot controller on the Intel Nuc computer. It collects all sensor information from the 14 modular servos and controls them in real-time. It hosts a high-speed feedback control loop (200Hz or above) to maintain robot posture and self-balancing. We refer to it as the low-level controller in Fig. 3.1.

The other edge controller is the robot command unit (RCU). It is a smart android phone with a private LTE connection and a 2D camera (Fig. 3.1). RCU serves as the gateway between the high-level cloud robotic platform and the low-level self-balancing controller. It uses the private LTE connection to stream live videos to the cloud. It receives and forwards high-level intelligent controls from the cloud to the low-level controller with minimum delay. RCU works both indoors and outdoors with a good LTE reception.

3.5.2 Cloud Controller

A high-level intelligent robot controller is placed in the cloud to work with the edge controller RCU (see Fig. 3.1). It operates at a lower speed (3-5Hz), yet it commands the robot based on HARI's Artificial Intelligence (AI) and Human Intelligence (HI) services, which is critical for robots to operate under unstructured environments. Depending on the situation, it can either extract commands based on the object recognition server or forward commands sent from a cloud teleoperator. These high-level commands are sent to RCU. They are then forwarded, transformed, and executed in the form of dynamic commands on the low-level controller.

3.5.3 Hybrid Control with "Heartbeat"

When controlling Igor, commands sent from the high-level cloud controller act as perturbations to a time-invariant, stable system maintained by the low-level self-balancing controller at the edge. This is a form of hybrid control where discrete signals are sent from the cloud to control a dynamical system at the edge.

To minimize network delays, we choose to use UDP, an asynchronous network protocol, to implement the Cloud-Edge communication. However, with UDP, packages can be lost during transmission. They can also arrive the designation out-of-order. Both problems can cause unstable, unpredictiv dynamic hybrid controls at the Edge, which can lead to bad user experiences and human safety issues during teleoperation.

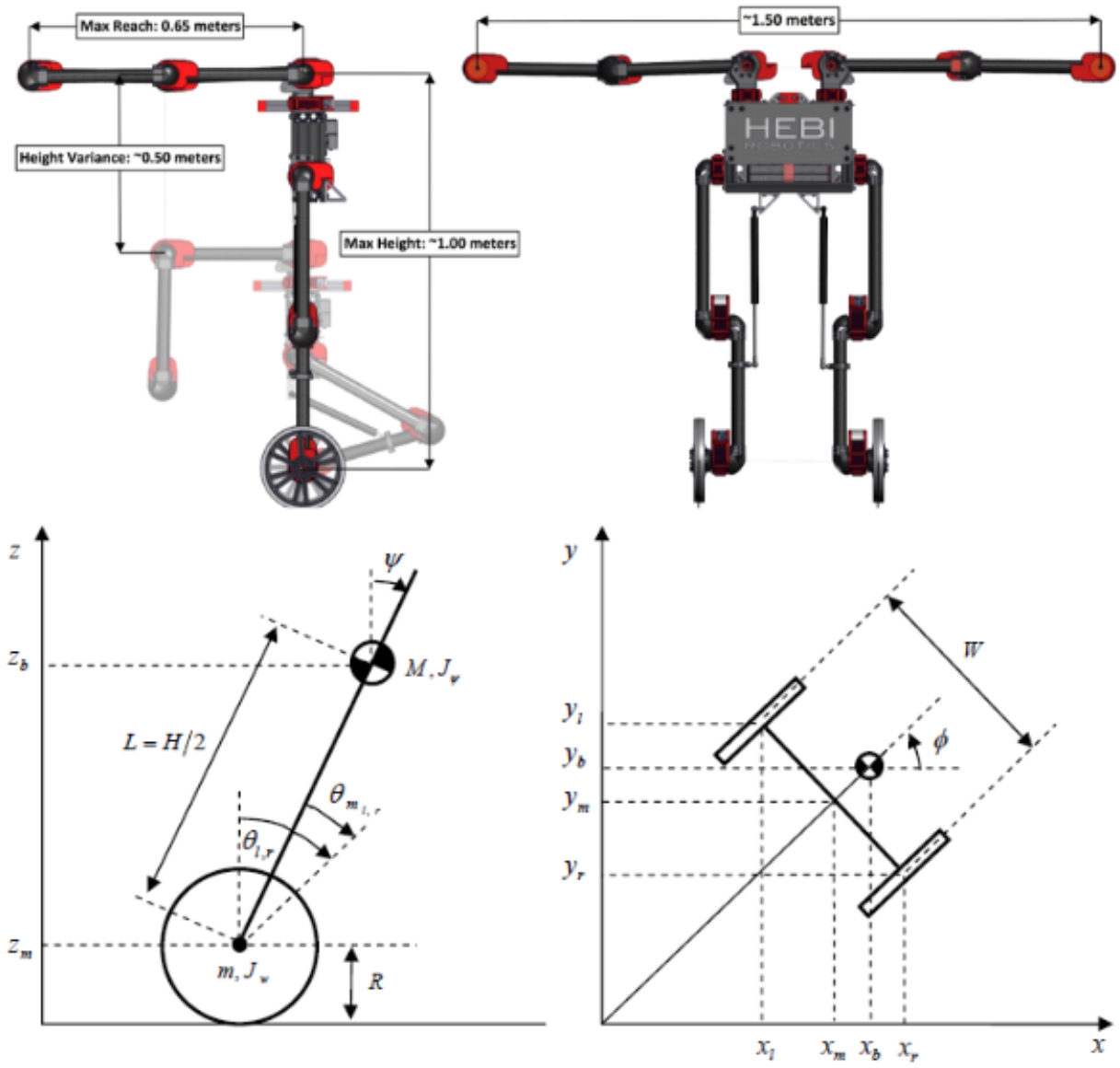

Figure 3.3: Self-Balancing Robot Igor: (Top)14 Dof, dual-arm, dual-leg robot built with series elastic servo modules. (Courtesy of Hebi Robotics) (Bottom) free body diagram of the inverted pendulum balancing control

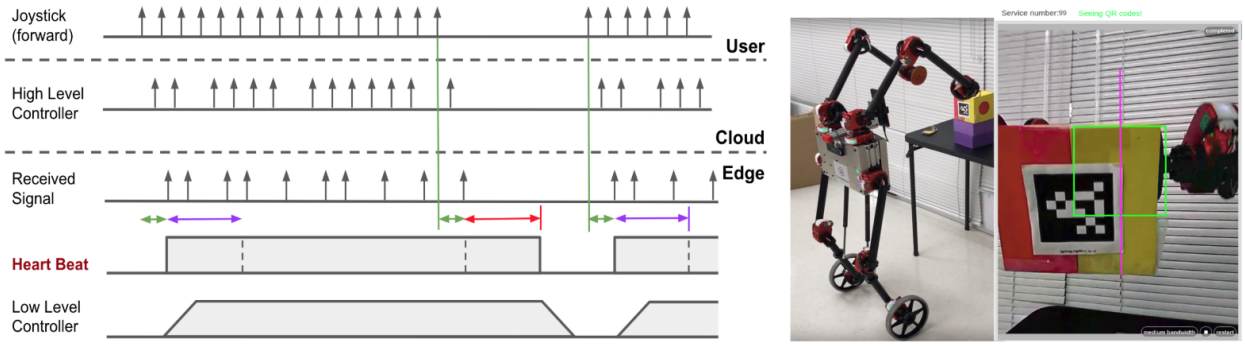

Figure 3.4: "Heartbeat" Asynchronous Protocol and Image Based Visual Servoing: (Left) "Heartbeat" Asynchronous Protocol: (Green) network delays from cloud teleoperator to robot Edge controller; (Purple) sliding windows that turn ON the "heartbeat"; (Red) sliding window that turn OFF the "heartbeat"; (Right) Image Based Visual Servoing (IBVS): Igor automatically picks up a Box using Fog Robotic IBVS. Videos: (1) Pickup from a table: https://youtu.be/b0mr5GHHjBg (2) Pickup from a person: https://youtu.be/K9R4y2w1uPw

We use a "heartbeat" design to solve these problems (shown in Fig. 3.4). The "heartbeat" is a switch signal that is turned on when the first control signal arrives at the Edge. It will remain on for a period of time (t) and will only turn off if there is no package received for the selected command during this time. We can view the "heartbeat" as performing a "convolution" with a moving window on the signal received from RCU. Finally, we add a ramping function at the beginning and the end of the "heartbeat" signal to ensure smooth and stable start and stop transitions.

3.6 Dynamic Visual Servoing

To assist teleoperation with automation, we use Fog Robotics IBVS to control Igor for automatic box pickups. We choose IBVS because it eliminates time consuming camera robot registration, which is hard to execute on a dynamic robotic system in an unstructured, human rich environment.

The goal of the IBVS is to navigate the robot to an optimal box pickup location where the apriltag lay exactly within the green target box (Fig. 3.4, right). To do that, we need to minimize the relative error between the measured target position and desired target position $\boldsymbol{e}(t)$:

$$\boldsymbol{e}(t) = \boldsymbol{s}(\boldsymbol{m}(t), \boldsymbol{a}) - \boldsymbol{s}^* \tag{3.7}$$

where $\boldsymbol{m}(t)$ is a set of image measurements and $\boldsymbol{a}$ is a set of parameters, such as camera intrinsics, that represents additional knowledge about the system. $\boldsymbol{s}$ is the measured values

of image features/object locations, such as pixel coordinates in the picture frame, and $\boldsymbol{s}^*$ is the desired values of image features/object locations.

The change of feature error $\dot{\boldsymbol{e}}$ and camera velocity $\boldsymbol{v_c}$ is related by *interactive matrix* $\boldsymbol{L}$:

$$\dot{\boldsymbol{e}} = \boldsymbol{L}\boldsymbol{v_c} \tag{3.8}$$

For IBVS, which is done in 2D image space, 3D points $\boldsymbol{X} = (X, Y, Z)$ are projected onto 2-D images with coordinates $\boldsymbol{x} = (x, y)$:

$$x = X/Z \tag{3.9}$$

$$y = Y/Z \tag{3.10}$$

which creates an interactive matrix for 2D image based servoing:

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

With the interactive matrix, camera velocity can be estimated by:

$$\boldsymbol{v_c} = -\lambda \boldsymbol{L}^+ \boldsymbol{e} = -\lambda \boldsymbol{L}^+ (\boldsymbol{s} - \boldsymbol{s}^*) \tag{3.11}$$

where L^+ is the Moore-Penrose pseudo-inverse of L:

$$\boldsymbol{L}^+ = (\boldsymbol{L}^T \boldsymbol{L})^{-1} \boldsymbol{L}^T \tag{3.12}$$

The final control law is set as a robot velocity effort v_s opposite to the camera velocity v_c because the target moves in the opposite direction of the camera in the image frame:

$$\boldsymbol{v_s} = -\boldsymbol{v_c} \tag{3.13}$$

Notice that the interactive matrix depends only on x and y, that is the 2D pixel coordinate of the target, and Z which is the depth of the target. In our system, Z is measured as the size of the apriltag. Therefore, the IBVS measurement is independent of the exact 3D position of the target measurement. This is a attractive feature for our system design because the exact 3D camera registration is not required for IBVS to complete the box pickups.

3.6.1 IBVS Implementations for Automatic Box Pickup

The automatic IBVS controller performs a box pickup in three phases. In phase one, the controller move the robot to a position where the aprialtag has the same size as the green box (Fig 3.4, right). The robot also need to position apriltag on the center purple line of the video frame after phase one, but not exactly at the center due to height differences. In phase two, the robot adjusts its own height by changing the joint angles of the two "knee" joints so that the apriltag would lay at the center of the video frame where the green box is. After the robot reaches the optimal picking position when apriltag is at the center of the video, phase three begins. The robot controller commits to perform a box-pickup with a pre-defined, hard-coded, dual arm, grasping motion.

3.6.2 Fog Robotic IBVS

Although we use simple apriltags for object recognition, we aim to anticipate the design of a deep-learning-based Fog Robotic visual system for robotic pickups. Therefore, to emulate the latency effects under such system, we deploy apriltag recognition in the cloud and use "heartbeat" protocol to stream apriltag's geometric informations–size and location–to low-level controller via RCU. Together, we build a robust, closed-loop Fog Robotic IBVS controller for box pickups.

3.7 Experiments and Results

With the "heartbeat" design, we can teleoperate the self-balancing robot reliably through the Cloud. To teleoperate the robot for box pickups, we hardcode a box pickup motion with the two robot arms. We first attempt to pick up a box with a joystick via direct cloud teleoperation. However, even with a reliable teleoperation module and pre-programmed pickup motion, we find it extremely difficult to finish the task using cloud teleoperation. We suspect that natural, immersive 3D visual perception is critical for a human to pickup objects efficiently, but a human teleoperator loses these perceptions using our current cloud teleoperation interface. In another word, our cloud teleoperation interface is not intuitive for efficient object pickups.

Table 3.1: Teleoperation vs. Automatic Box Pickups

	Average Duration (s)	Success Rate
Local Teleop	43	9/10
Cloud Teleop	340	4/10
Auto Pickups	46	10/10

To quantify the observation, we perform two different teleoperation experiments with 10 trials each: (1) control Igor locally so that the operator is close-by and can see the robot and the box; (2) control Igor remotely from the cloud to pickup the box through the teleoperation interface (Fig. 3.4, right). In both cases, the box is positioned at the center of the table. The robot is 2 meters from the object and faces the front of the box (see setups in video https://youtu.be/b0mr5GHHjBg). We observe that local teleoperator can perform box pickups much faster with a higher success rate than the cloud teleoperator (see table 3.1)

To assist the cloud teleoperator for improved intuition and efficiency, we implement the automatic IBVS module. We benchmark the same experiments using the automatic module with human in the loop. In these experiments, we allow the cloud teleoperator to first teleoperate the robot as fast as possible to a location where apriltag is recognizable (up to 20 degrees from the surface normal of the apriltag) and is about 2 meters away. The

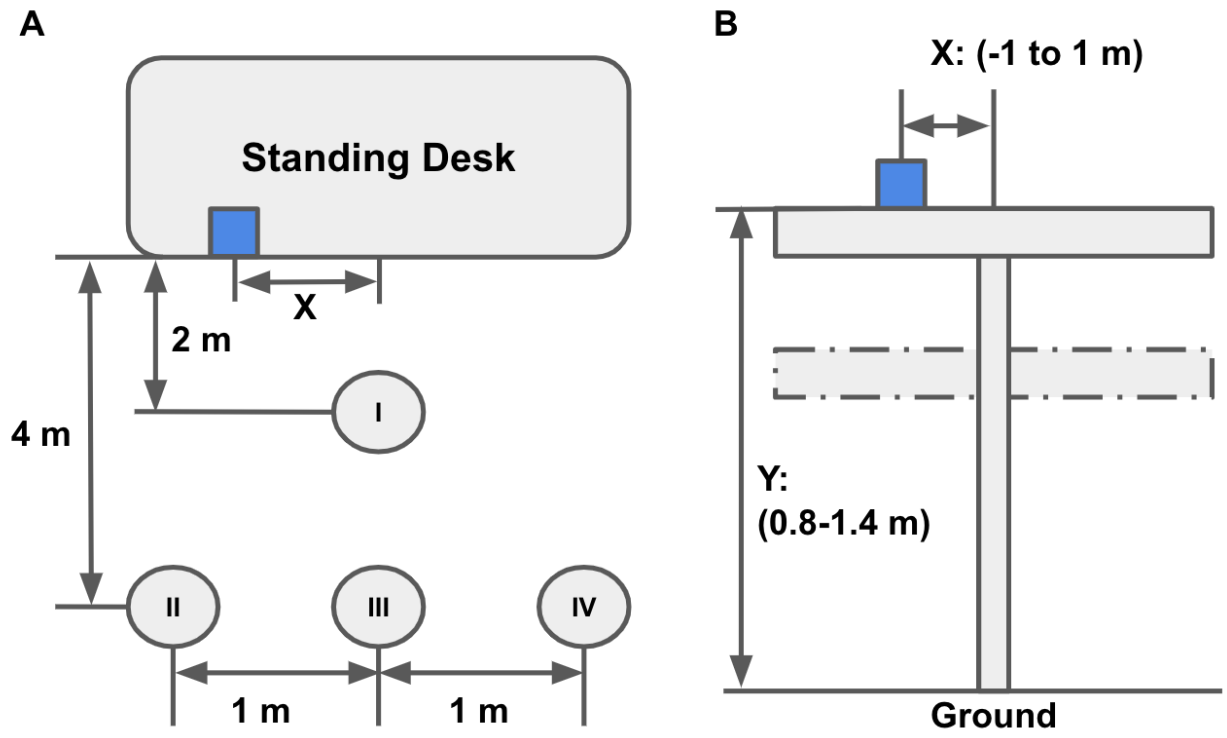

Figure 3.5: Igor Box-Pickup Reliability Tests: (A) Top down view: four starting positions in respect to the standing desk; (B) Front view: The position of the box is generated randomly from two uniformly distributed variables, horizontal coordinate X, in range (-1, 1) meters, and vertical coordinate Y, in range (0.8, 1.4) meters.

teleoperator then switch on the Fog Robotic IBVS module, allowing the robot to pick up the box automatically. We observe that the speed of this approach is on-par with human local teleoperation, but the reliability is even higher at 100% (see table 3.1)

Table 3.2: Successful Automatic Pickups of a Randomly Positioned Box

Start Location	Point I	Point II	Point III	Point IV
Success Rate	8/10	9/10	9/10	10/10

We further conduct 40 reliability trials. At each trial, Igor start from one of four starting locations–Point I-IV (Fig. 3.5, A) to pickup a box that is put at a random position on a standing desk. We perform 10 trials for each robot starting location, and the box position is defined by two uniformly distributed random variables–horizontal X coordinate in the range

of -1 and 1 meters and vertical Y coordinates between 0.8 and 1.4 meters (Fig. 3.5, B). We show high reliabilities of the automatic, IBVS box-pickup system through these trials in table 3.2

Finally, we use the IBVS module assist a cloud teleoperator to pickup a box efficiently from a box carrier. During this task, a moving human carrier is holding a box with an apriltag. A cloud teleoperator operates Igor to a location where the apriltag can be recognized. As soon as the robot recognizes it, the teleoperator would swtich the robot into IBVS mode so that the robot can start automatic box pickups. We observe that as long as the robot can track the apriltag, it will dynamically follow the human around; and if the human carrier stops and holds the box at a static reachable position, the robot will eventually pick up the box from the carrier.

3.8 Discussion and Future Works

In this work, we combine intelligent Cloud platforms and fast Edge devices to build a Cloud-Edge hybrid Fog Robotic IBVS system that can perform dynamic, human-compliant, automatic box pickups. A "heartbeat" protocol with asynchronous communication is implemented under this framework to mitigate network latencies and variabilities. However, the current "heartbeat" protocol is not perfect. The "heartbeat" actually increases delay at the end of the action, which would cause further delayed reaction after the last command signal. This is a significant safety concern, because even if the "heartbeat" time window is short, i.e. 250 ms in our case, the robot will continue to move until after the 250 ms window. To compensate, we implement a sharper ramp function at the end-of-action, but 250 ms is the hard limit for this kind of delay with the current "heartbeat" system (red arrow in Fig. 3.4).

In the future, we can further reduce such delay by modeling variabilities of time intervals between package arrivals due to package loss. This way, the Edge controller can predict when the last package would arrive, so that it can finish the action before the arrival of the last command.

Like other service robots, Igor needs to interact and cooperate with human beings. We demonstrate the advantages of visual servoing: (1) it requires no calibrations before each robotic task; (2) it can handle dynamic human robot interaction, such as following a human to pick up a box from that person. One failure case is when the human carrier tricks the robot. It happens if the human carrier move the box after the robot commits to the final phase of box picking, which is hard-coded. Instead, we can automate the pickup motion based on visual servoing as well, and we will leave it as future work.

For Fog Robotic IBVS, we emulate a cloud-based deep-learning visual perception system by deploying apriltag recognition in the cloud. As part of future work, we plan to deploy deep-learning recognition pipelines such as mask-RCNN [50] together with intelligent grasping systems such as dex-net [88][89][83], so that it can guide both dynamic robots such as Igor and static robots such as YuMi[129] and HSR [77] to perform generalized, human compliant object pickups and manipulations.

Notes

This chapter has appeared in a peer-reviewed conference paper:

Nan Tian, Ajay Kumar Tanwani, Jinfa Chen, Mas Ma, Robert Zhang, Bill Huang, Ken Goldberg and Somayeh Sojoudi. *A Fog Robotic System for Dynamic Visual Servoing.* ICRA 2019.

Chapter 4

A Cloud-Edge Hybrid System for Humanoid Gesture Imitation

4.1 Introduction

Humanoid social robots are available commercially, including Softbank's Pepper [106] and Nao [105], AvatarMind's iPal, and iCub from RobotCub [94]. These robots have similar appearances as humans, and have potentials in professional services, such as retail, hospitality, and educations. Pepper, a humanoid social robot designed by Soft-bank [106], is well liked because it has a human voice, Astro boy liked body design, and generous full body movements. It is also well equipped with cameras, LIDAR, ultrasound sensors, and a chest input pad, integrated either through Pepper's android SDK or ROS based system.

In this work, we focus on using Pepper to mirror a person performing semaphore, a hand-held flag language. It is a system conveying information at a distance by visual signals with hand-held flags. Alphabetical letters are encoded by the positions of the flags with respect to the body. It is still used by lifeguards around the world.

To perform semaphore recognition, we use 2D video stream that is common in low-cost humanoid robots. However, it is impossible to deploy a real-time gesture-based recognition system locally on humanoid robots due to limited computation power. Deep learning based gesture recognition systems such as OpenPose [23] can only achieve real-time performance on state-of-the-art deep learning GPU servers. At the same time, compared to Microsoft Kinect based gesture recognitions [13], OpenPose is preferable as it can work outdoors, can track more people, and is camera agnostic. To use OpenPose for robot gesture mirroring in real-time, one option is to stream videos from local robots to the cloud to perform OpenPose gesture recognition and send control signals back to local robots.

To illustrate this concept, we use CloudMinds Inc. HARI, Human Augmented Robot Intelligence, to support large-scale cloud service robot deployment. HARI is a hybrid system that combines the power of artificial intelligence (AI) and human intelligence to control robots from the cloud.

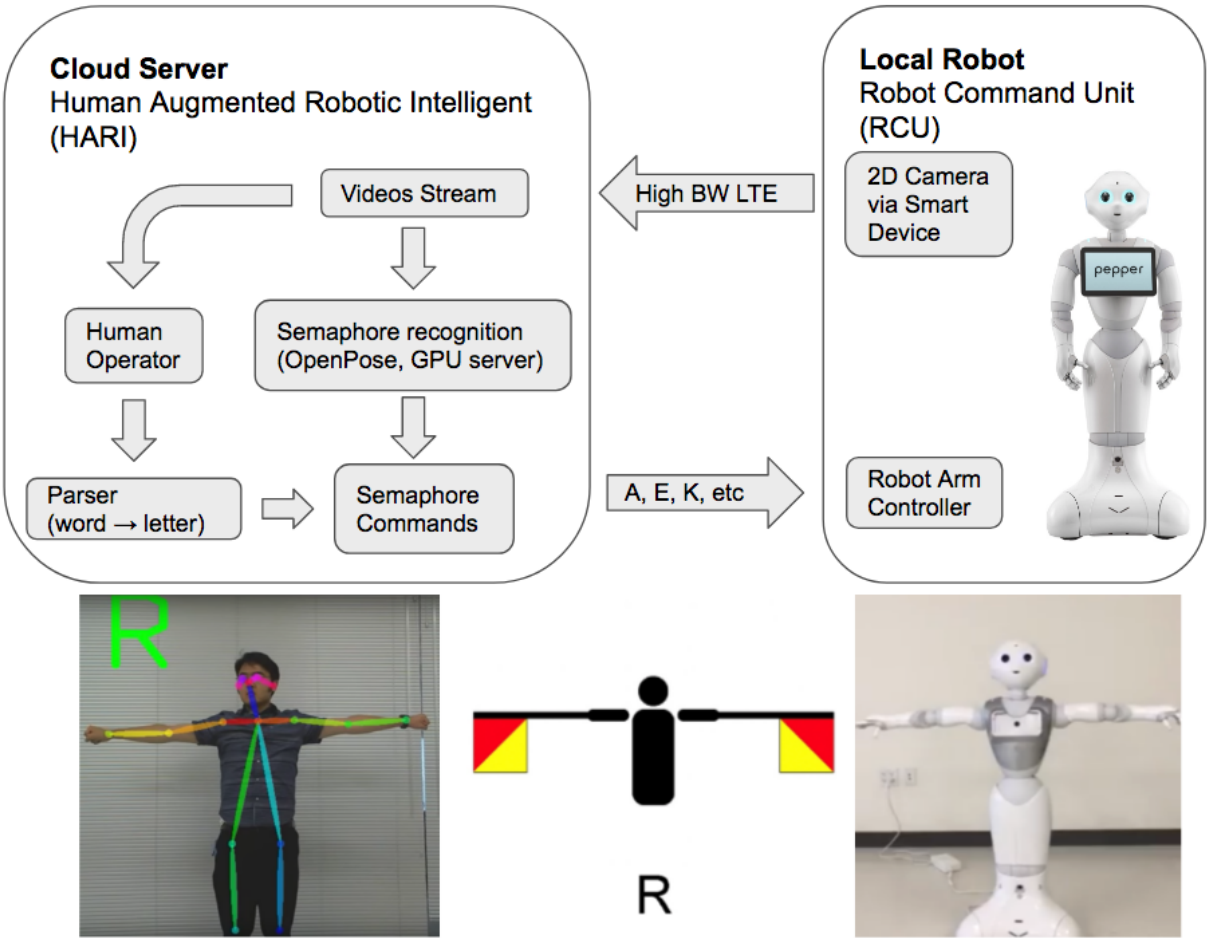

Figure 4.1: Cloud-Edge 'Hybrid' System for Semaphore Imitation Block diagram: (Top) Architecture diagram of the cloud-based semaphore mirroring system built on HARI, which supports AI assisted teleoperation. We deploy semaphore recognition system in the cloud, and used a discrete-continuous hybrid control system to control Pepper, a humanoid robot, to imitate semaphore gestures. (Bottom) An example of our system when the demonstrator performs semaphore "R", and the robot imitate him

Mitigating network latency is essential to make cloud-based motion control robust, because network latency is unavoidable at times due to routing and physical distance even with a reliable network. We pre-generate semaphore motion trajectories and store them in the local robot command unit (RCU). We execute these trajectories based on commands sent from the cloud. This way, the cloud AI only needs to use alphabetical letters to control robots to perform semaphore, which hides the network latency while the robot moves from one semaphore to another.

To test network speeds, we conducted cross-Pacific experiments to control a Pepper robot in Japan from a HARI server located in the United States while the human subject was standing in front of the robot in Japan. We further showed that one HARI server could control two robots located in two different locations simultaneously, which demonstrates scalability.

4.2 Related Work

An anthropomorphic robot, or humanoid robot, refers to a robot whose shape resembles the human body. Many famous robots are humanoids, including Atlas from Boston Dynamics [76], Honda research's Asimo [52], and the first space humanoid robot–NASA's Robonaut 2 [28], just to name a few.

Socially interactive robots [35], though not necessarily anthropomorphic, often take humanoid forms, since many researchers believe humans must be able to interact with robots naturally, or as human like as possible [55, 56]. Others find them easier to control by emulating natural human robot interactions [35]. Therefore, many works have been dedicated to imitating human gestures [20, 21, 19, 116, 101], human demonstrations [4, 19] and human social interactions [17, 16] using humanoid robots.

Imitating full body human gesture requires a 3D positioning camera like Microsoft Kinect and a gesture recognition system such as Kinect SDK [13] or deep neural network based OpenPose [23]. The imitating controller maps position and orientation estimation of human joints into a series of robot joint angles–trajectory. These generated gesture trojectories are then used to control humanoids directly [13], indirectly after editing [135], or with a model learned from machine learning [20, 19].

In Goodrich's HRI survey [43], he separates HRI into two types: remote interaction and proximate interaction. In either case, perception feedback–visual, audio, verbal, or even tactile–is important to control and program robot for interaction. In this work, we focus on large-scale cloud-based social human robot interaction using a humanoid robot, named Pepper [106]. We enabled both remote interaction–teleoperation–and proximate interaction by deploying both visual perception–gesture recognition, and high-level robot command–semaphore–in the cloud. A similar cloud robotic system was built by Kehoe, et. al. before [67] when an object recognition system was deployed in the cloud to support the PR2 robot grasping system.

James Kuffner coined the term "Cloud Robotics" to describe the increasing number of robotics or automation systems that rely on remote data or code for effective operation [75]. A wide variety of models for connecting robots to the cloud have been developed [65]. Some studies have used an SaaS model that moves a robot motion controller entirely into the cloud and used it for manipulation planning [133], while others, such as RoboEarth's Rapyuta system [95], follow a Platform as a Service (PaaS) such that others would easily program on the platform. A third kind of cloud robotic system, aiming for a more energy efficient robot system, distributes the computation of robot motion plannings to both robot's embedded computer and a cloud-based compute service [57]. HARI is a combination of the three and aims at providing both AI and human intelligence control for large deployment of service type robots.

4.3 System Design

We describe three basic components of our cloud-based semaphore mirroring system: humanoid robot, robot command unit, and cloud intelligent HARI; and explain how cloud-based semaphore system was implemented, including semaphore motion generation in ROS, trajectory execution at RCU, command line interface on HARI, semaphore recognition, and semaphore mirroring with hybrid control.

4.3.1 Humanoid Robot Pepper

We use Pepper, a humanoid social robot designed by Soft-bank, for all implementation. Pepper has two arms. Each arm has five degrees of freedom (DoF), with one more degree of freedom for hand closing and opening. Further, Pepper has two DoFs for head movements, two DoFs for hip movements, and one additional DoF for knee movements.

The robot is natively controlled through an on-board Atom E3845 CPU card. This is where robot sensors and motion commands are integrated. An experimental ROS node is available to control Pepper directly via the CPU over a wireless network.

4.3.2 Robot Command Unit (RCU)

The Pepper robot has a smart input pad with an Android based system on its chest. We refer to this smart device as robot command unit (RCU). The smart device communicates with the CPU to integrate sensor information and send robot commands via hardwired connections. It serves as the primary central control of the robot. The robot manufacturer allows the user to program the RCU directly with Android API. The user can alternatively program Pepper from another PC via WiFi using ROS, which can be convenient for roboticists.

However, we chose to directly program the robot semaphore execution on RCU rather than over WiFi with ROS for the following reasons. First, a hardwired connection is fast, reliable, and free from any interruption caused by poor WiFi connections. Further, the smart

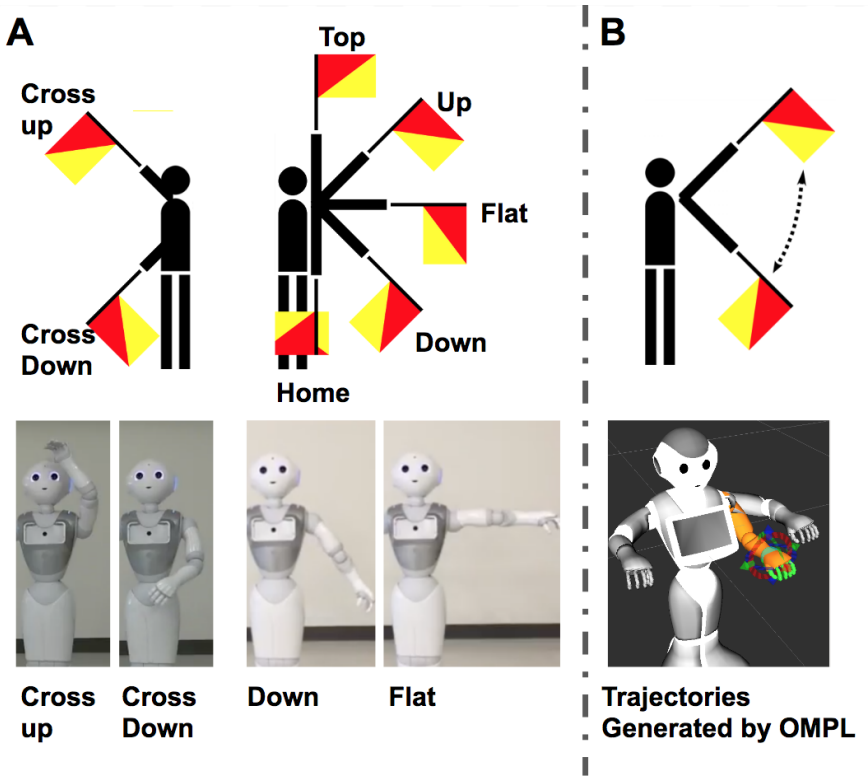

Figure 4.2: Programming Single Side Semaphore: A. Semaphore positions (Top) Seven different positions–cross up, cross down, top, up, flat, down, home–demonstrated with cartons. (Bottom) Selected examples of single side semaphore positions implemented on Pepper. B. Semaphore Motion Motion trajectories in between different semaphore positons are generated first using MOVEIT! and OMPL in ROS, and then stored on local RCU.

device has wireless connectives, for both indoors or outdoors using WiFi or mobile LTE, and it can act as a natural connection to the cloud services. Additionally, touch-screen-based GUIs are easy to use for local users. Finally, onboard sensors, especially cameras, can be used as perception inputs for the robot. Therefore, we position the RCU as a local robot controller as well as the local network gateway from cloud robotic services. This is illustrated in Figure 4.1.

4.3.3 Cloud Intelligence HARI

Currently, HARI has three major modules: (1) cloud-based AI engine and HI teleoperation interface, (2) RCU in the form of smart devices as WiFi and mobile LTE gateway to cloud services, and as mid-level controller for local robot, and (3) a high bandwidth, reliable, secure private-LTE network connecting the cloud AI directly to local robot via RCU (see figure 1).

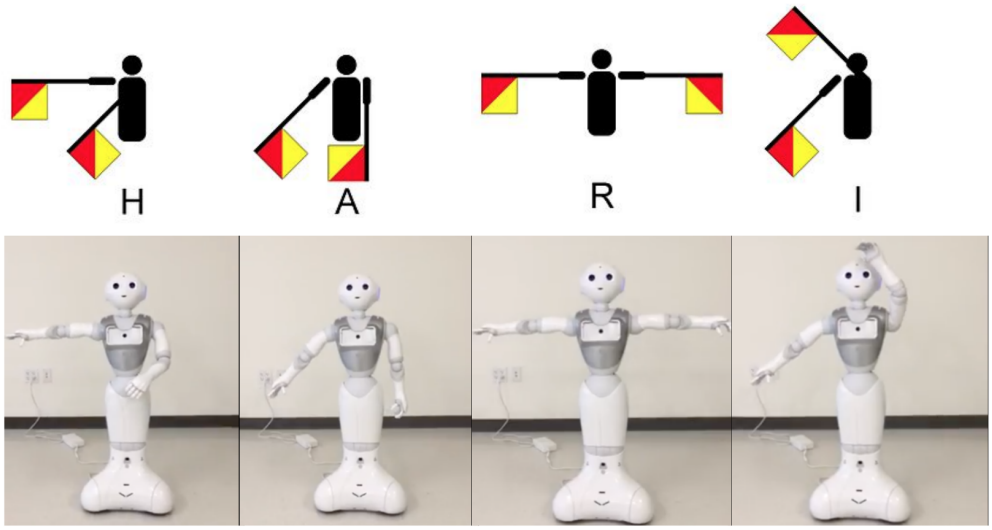

Figure 4.3: Selected Example of Semaphore Performed by both Arms of Pepper Combination of the seven positions of each arm can be used to generate the full alphabet with 26 letters. The "HARI" sequence was teleoperated via command line interface on HARI, (video: https://youtu.be/XOeiDrG4sAQ)

As mentioned before, the key idea in building a real-time, scalable cloud application with wide geological coverage is to maximize the cloud computation for valuable AI services and minimize the communication costs for each robot via RCU. We use semaphore mirroring as an example to demonstrate how use these ideas to build such cloud-based applications in real life. First, we move the deep learning based gesture recognition system, OpenPose, entirely to HARI's cloud AI GPU servers. The GPU server includes four Nvidia Titan Xp graphics cards. Second, to minimize the communication costs, we stream 320x320 videos with highly efficient video compression standard H.264 from RCU to Cloud HARI through private LTE.

Third, mitigating the network latency is important to ensure a robust motion control over long-range communication, because network latency is unavoidable at times due to routing and physical distances even with a reliable LTE network. Therefore, we store pre-generated semaphore motion trajectories in the local RCU, and execute these trajectories based on commands sent from the cloud. This way, we hide the network latency caused by video streaming and cloud control during the trajectory execution when the robot is moving from one semaphore to another. (Fig. 4.2). Additionally, the cloud AI only needs to send intermittent alphabetical letters to control robots to perform semaphore. Please refer to Section IV and V for an exposition of these three design decisions.

4.3.4 Semaphore Motion Generation in ROS

Semaphore is the telegraphy system conveying information at a distance using visual signals with hand-held flags. Flags encode alphabet information with positions of both arms. Each of the two arms has the same set of different arm positions: home, down, flat, up, top, cross down, and cross up (see Fig. 4.2A). The combination of these seven positions using two arms forms 49 possible stationary positions. These positions are used to represent the alphabet (Fig. 4.3A), and additional symbols such as home, space, etc. (Fig. 4.4C, 4.4D).

Based on the above definition, we first program the seven different positions of each arm for the Pepper robot (fig. 4.2A). We then use the Pepper's ROS interface to generate trajectories that would move each of the Pepper's arms between any two of these seven positions. (fig. 4.2B) We use SBL, short for Single-Query Bi-Directional Probabilistic Roadmap Planner [110], to generate these trajectories in MOVEIT! using OMPL. We record the resulting trajectories to ROS bags, and convert them into Android readable format.

4.3.5 Trajectory Execution at RCU

We store pre-generated trajectories in RCU. Upon receiving an alphabetical letter command, RCU on Pepper would look up the trajectory that corresponds to the current hand position as start and the received semaphore letter position as goal. It will then execute this trajectory locally until the robot arms stop at the expected commanding positions. This is programmed using Pepper's Android API.

Note that the execution is a blocking function. The robot arm would not change its execution trajectory or take any other command until the motion is finished. We later take advantage of this feature to hide the network latency for responsive user experiences.

4.3.6 Command Line Interface on HARI

We test this local robot semaphore controller through a command line interface in HARI. In this interface, a cloud operator would input a letter command via HARI. When RCU receives the letter, it performs semaphore as described above.

We further modified this command line interface so that a word or a sequence of letters can act as a command to Pepper. When a word is sent, Pepper will perform semaphore for each of the letters in sequence. It would also spell out the letters as it moves and pronounces the word at the end of the sequence. Fig. 4.3 illustrates Pepper performing "HARI" in sequence. This demonstrates a form of AI-assisted teleoperation in which a cloud operator can communicate to the person in front of the robot in the form of combined semaphore motion and spoken word by using Pepper as the media. (see supplemental video for a demonstration: https://youtu.be/XOeiDrG4sAQ) This interface becomes a part of HARI teleoperation module during the developments (Fig 4.1Top).

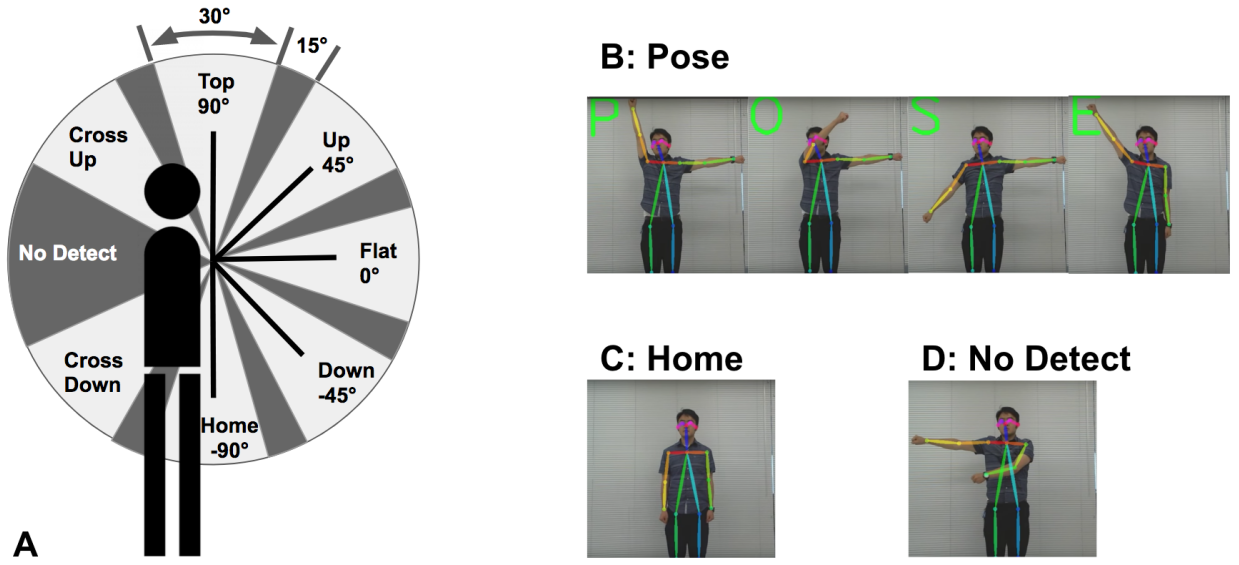

Figure 4.4: Semaphore Recognition using Openpose: A. Single Side Position Recognition Seven positions are recognized based on a unit circle centered at the shoulder joint; the angle between the line formed by the hand and the shoulder and a horizontal line is used to recognize the seven positions of a single arm. Light zones indicate the detection zones, whereas dark zones are "No Detect" zones. No detect zones are used to avoid false detection between two neighboring positions. We can reliably recognize the full semaphore alphabet in real-time (video: https://youtu.be/arC6OZGgkgE). B. the user spells out "POSE" and the semaphore recognition system recognizes the letters. C "Space" is represented by both arms rest in the "home" position. D. "No Detect" is recognized because the left arm of the demonstrator is in the dark "No Detect" zone.

4.3.7 Semaphore Recognition

Semaphore mirroring is built on top of the OpenPose based semaphore recognition system and the command line teleoperation system mentioned above. To program semaphore recognition, we use only the largest skeleton extracted from OpenPose. This skeleton corresponds to the person standing in front of the robot. All other skeletons in the background are filtered out. Based on this skeleton, we measure the relative angle from the left-hand position to the horizontal line crossing the right shoulder. If this angle falls into the perspective gray detection zone (Fig. 4.4A), then the left arm's semaphore position is recognized. Note that the black area indicates a "no-detect" zone. We setup the no-detect zone to prevent false detections between the two neighboring positions since the skeleton extracted by OpenPose can be noisy and the detected hand position can easily cross the border at times.

The right-hand semaphore recognition is programmed the same way, except that the

detection zone map was horizontally flipped from the one illustrated in figure 4.4A. Combination of the detected left-and-right-hand position is used to infer semaphore based on its alphabet. In Figure 4.4B, we show that the system could detect multiple letters of the alphabet, and the subject spells out the word "POSE" using semaphore. Figure 4.4C shows the home position which corresponds to a "space" in the alphabet, and Figure 4.4D shows a case of "no-detect" because the left hand of the subject is in the "no-detect" zone.

4.3.8 Semaphore Mirroring with Hybrid Control

When the cloud AI detects a change in the human gesture in semaphore, HARI will use the command line interface to send a letter to Pepper's RCU. RCU takes this letter command and move the robot arms to that semaphore position. This completes the semaphore mirroring pipeline. To minimize communications, HARI will not send in semaphore command at every frame of semaphore detection. It only sends a command to RCU when it detects a change in semaphore letter recognition, see fig. 4.5.

However, there is a problem if we use the raw command signal sequence sent from HARI. The semaphore motion executed by RCU is a blocking function. The RCU would not process the command buffer during the motion. If we choose to program the receiving buffer at RCU using a stack, the original command sequence were to be reserved. So when multiple semaphore changes are to be detected during the semaphore motion execution, then the letter representing the "first" detection are executed first, then followed by the other ones, until the most "current" semaphore command is detected. It surprises many users as the robot executes several semaphores before it starts to execute the semaphore they are intended to demonstrate.

An alternative is to program the receiving buffer as a queue, so that when a motion was finished, RCU jumps to the most current semaphore gesture demonstrated by the user. Users would feel more natural since they perceive the robot is mirroring them faithfully. Therefore, the queue-based receiving buffer is chosen.

The semaphore mirroring pipeline is a form of hybrid control–Discrete, high level command is sent from the cloud intermittently to a local RCU where continuous motion is executed. As we will discuss later, this technique helps hide latency between the cloud and the robot. It makes the cloud-based real-time semaphore mirroring Pepper more natural to interact with, even under extreme network conditions.

4.4 Experiments and Results

4.4.1 Cross Pacific Experiments

We conducted cross Pacific robot control experiments. We used a US HARI server located in Oregon, United States to control a Pepper robot located in Tokyo, Japan, with a human subject standing in front of Pepper as a demonstrator. We also performed the same test to

a Pepper robot located in Santa Clara, United States, with the same HARI server in the United States (see video illustrates US-to-US test case: https://youtu.be/NXrdoSXXxqY). Furthermore, to show scalability, we successfully used the US based HARI server to support both Peppers at the same time, one in Japan and the other in the United States.

4.4.2 Reliability of the Cloud Semaphore Recognition System

We first performed reliability tests of semaphore recognition system in the United States. To test reliability, we randomly showed a semaphore position for the demonstrator to perform, and check the recognized semaphore results obtained from the cloud server. We conducted two sets of experiments with two different demonstrators with thirty experiments each. We positioned the robot camera in such a way that both shoulders of the subject were at the center horizontal line of the image. Further, no arm, when fully stretched, should go out of the camera's frame. The background of the demonstrators were uniformly white, and the experiments were conducted in a well-lit room.

During the first set of experiments, we hid the real-time recognition results from the demonstrator. The recognition accuracy was 90.0% (27/30) for subject one, and 76.7% (23/30) for subject two. (see the first row of table 4.1) The accuracy were high, but not reliable, and not consistent across subjects.

However, we noticed that all of the failures were caused by "No Detect" error, meaning one or both of subject's hand was in the no detect zone. None of these recognition errors was due to confusion between two different semaphores. This made us think that the failed trails were caused by habits of different person when performing semaphore rather than system errors such as a high noises level of OpenPose's recognized joints positions. The actual recognition accuracy can be significantly higher if the demonstrators were trained, or if the real-time results were shown to the demonstrators.

Therefore, we conducted a second set of experiments for the same subjects by allowing them to look at the semaphore recognition screen to adjust their pose for a better recognition result. However, they could not adjust their arm so that their arms moves outside the quadrant they were in. A quadrant was defined as the uni-circle made by the vertical and horizontal lines in these experiments.

We obtained perfect accuracy of 100% for both subjects in the second set of experiments. (see the second row of table 4.1) Therefore, it can be concluded that the semaphore recognition system is highly reliable for the purpose of remotely teleoperating a robot to perform semaphore mirroring (see video demonstration: https://youtu.be/arC6OZGgkgE).

4.4.3 Real-Time performance of the Cloud Semaphore Recognition System

We used semaphore mirroring as an example to demonstrate the benefit of using a cloud-based AI platform. We measured the real-time performance of the cloud-based OpenPose

Table 4.1: OpenPose Semaphore Recognition Accuracy (%)

	Subject 1	Subject 2
Open Loop	27/30	23/30
with Feedback	30/30	30/30

system. We used a single Titan Xp GPU with a single thread to launch OpenPose in the cloud. We fed a local video stream with resolutions of 640x480 to OpenPose. We observed 10-12 fps (frames per second) inferencing framerate while using a pre-trained 320x320 network provided by OpenPose.

We then measured the communication costs to stream video to HARI. To minimize communication costs, we streamed in a 640x480 resolution with a highly efficient video compression standard H.264 from RCU to Cloud HARI through private LTE. We measured consistent 22-30 frames per second video stream both from the US-to-US test case and from Japan-to-US test case. The difference between the two cases are video streaming latency. There is around 100 ms latency for the the US-to-US case, and around 400 ms latency for the Japan-to-US case (Table 1)

This suggests that semaphore recognition is the bottleneck for the cloud-based recognition system, which still provides 10-12 fps real-time performances even if we stream video from Japan to the US. The cloud-based OpenPose system has an order of magnitude better performances compared to OpenPose on an iPad 2017, which can only process a little more than one frame per second or less (cite swiftOpenPose).[23].

The results from these experiments support our claim that communication costs are lower compared to the benefits gained from using a cloud robotic AI framework, in our semaphore mirroring system.

4.4.4 Hybrid System Design Hides Network Latency

A hybrid system can hide the network latency in the robot motion in this cloud-based system. The mechanism is illustrated in Fig. 5. The semaphore motion time average is 3.4 seconds. It is long compared to the total latency contributed by video streaming and robot commanding (Fig. 5B). After the demonstrators are used to the system, they tend to start the next semaphore demonstration before the robot finishes executing the last semaphore. The semaphore recognition system in the cloud would recognize any change of gesture earlier despite the delay from video streaming. The change in gesture would trigger HARI to send the next semaphore command, which can be delivered to RCU before the end of the last semaphore execution as well. Therefore, the next command would sit in RCU's receiving buffer until the end of the last motion, and the is executed immediately.

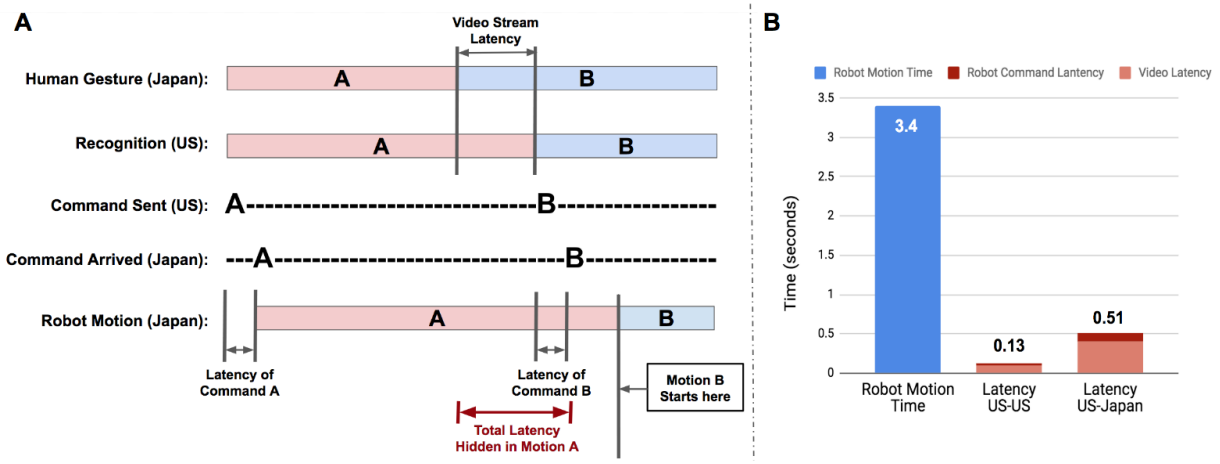

Figure 4.5: A Hybrid System Hide the Network Latency: A. Timing Diagrams of the Hybrid System The diagram shows that, starting from the actual human gesture in Japan, the video latency and latency of sending in a command after recognition are consecutive. The sum of the two forms of latencies is marked in red. The total latency is hidden inside motion A, but occurs before the start of motion B. B. Timing Comparison Robot semaphore execution time is much higher than either the US-to-US network latency or the US-to-Japan network latency.

Table 4.2: Network Latency Breakouts (ms)

	Video	Robot Command	Total
Japan to US	400	108	508
US to US	98	31	129

4.4.5 Compared to a Local Full Gesture Imitation System

We also built a general gesture imitation system for Pepper using a 3D Kinect camera with a local PC in ROS (see video: https://youtu.be/fPqF1xwqRcY. To compare this local general gesture imitation system and our cloud-based semaphore mirroring system, we performed semaphore mirroring using both systems.

The general gesture imitation system finishes a semaphore within 5-10 seconds which is slower but close to the performance of our cloud-based system. However, it failed when the human demonstrator drove Pepper into unreachiable areas in its arm. This failure cases happens more often when the start and goal positions of the robot is far away, for example

from home to top, or when any of the cross positions are involved. In these cases, Pepper would stuck in those places for a while and the user would need to find a way to drive the arm out via demonstration. There are also times when the user drives two arms to collide to each other, though this does not happen often during semaphore demonstrations. In contrast, the cloud-based semaphore system never failed because the trajectory was generated from an automatic path planning algorithm. The system checks for kinematic constrains and self collisions during the generation. We will discuss the trade-offs between the two systems next.

4.5 Discussion and Future Work

Real-time visual perception with cloud computing is difficult, because the network bandwidth requirement is very high compared to ASR and NLP. We showed that, with a private-LTE network and a highly compressed video stream, we can achieve a 22-30 fps video streaming rate even if we stream from Japan to the United States. This is faster than the cloud-based gesture recognition rate–10-12 fps. This confirmed our hypothesis that the value of running a more powerful AI service in the cloud would reduce the communication costs, as even better and cheaper communication technologies, such as 5G LTE network, would become available in the future.

In the HARI cloud-based hybrid system, discrete command is sent from the cloud and the robot performs per-generated motion upon receiving the command. We only send a semaphore command when there is a change in semaphore recognition, makes the communication highly compact. And as we illustrate in Fig. 5A, the hybrid system help hide network latency during robot motion, making the human robot interaction responsive even if the cloud server is far away.

There are, however, limitations of this hybrid, cloud based, gesture imitating system. It can only imitate a discrete set of 2D gestures, semaphore, which is a sub-set of general human gesture. It also require roboticist to program and generate these motions before-hands and store them in the RCU, which can be labor intensive if more general gestures are needed.

As future work, we aim to make the semaphore mirroring system more general. It is desirable to create discrete gesture segments from a large human activity database using machine learning techniques, with obviates the needs for explicitly programming poses and gestures by hand. Furthermore, it is important to explore human robot interaction rather than gesture imitating.

Notes

This chapter has appeared in a peer-reviewed conference paper:

Nan Tian, Benjamin Kuo, Xinhe Ren, Michael Yu, Robert Zhang, Bill Huang, Ken Goldberg and Somayeh Sojoudi, *A Cloud-Based Robust Semaphore Mirroring System for Social Robots.* CASE 2018.

Chapter 5

Mitigate Network Latency using Motion Segmentation and Synthesis

5.1 Introduction

Cloud Robotics connects robots with limited computation power to the Cloud through the Internet. It enables these distributed robots to access computation intense machine learning (ML) modules in the Cloud. One important application of Cloud Robotics ML modules is tele-operation via shared autonomy. It often requires a closed-loop controller to react to human action feedbacks interactively in real-time.

Network latency is one major problem in Cloud Robotics and long-range tele-operations. It is caused by propagation delays and network routings delays which can be highly variable and unpredictable. This imposes challenges to build a reliable cloud-based real-time closed-loop controller to tele-operate dynamic robots, because variable delays in the feedback communication can lead to uncontrollable oscillations, which are unsafe for human rich environments. Further, an unpredictable lag in response to a human action can cause counter-intuitive human robot interactions, which would lead to sub-optimal user experience.

One way to mitigate latency is to hide network latency inside robot motion executions. To do that, we need an ML based closed-loop controller that can recognize and predict where the tele-operator would go. Based on these predictions, the robotic controller should synthesize and execute motion segments with similar characteristics. Such shared autonomy system can help the remote robot move to intermediate or final targets on time, eliminating network delays.

To demonstrate, we prototype a tele-operation system that enables a remote human to draw handwritten letters using a dynamic robot. (1) We learn a dictionary of motion segmentation HSMM models for each handwritten letter; (2) we share these models with both the tele-operator interface and the robot edge controller; (3) the system remotely command the robot to execute these segments in response to human demonstration using HSMM; (4) the controller synthesize motion segments with linear quadratic tracker (LQT)

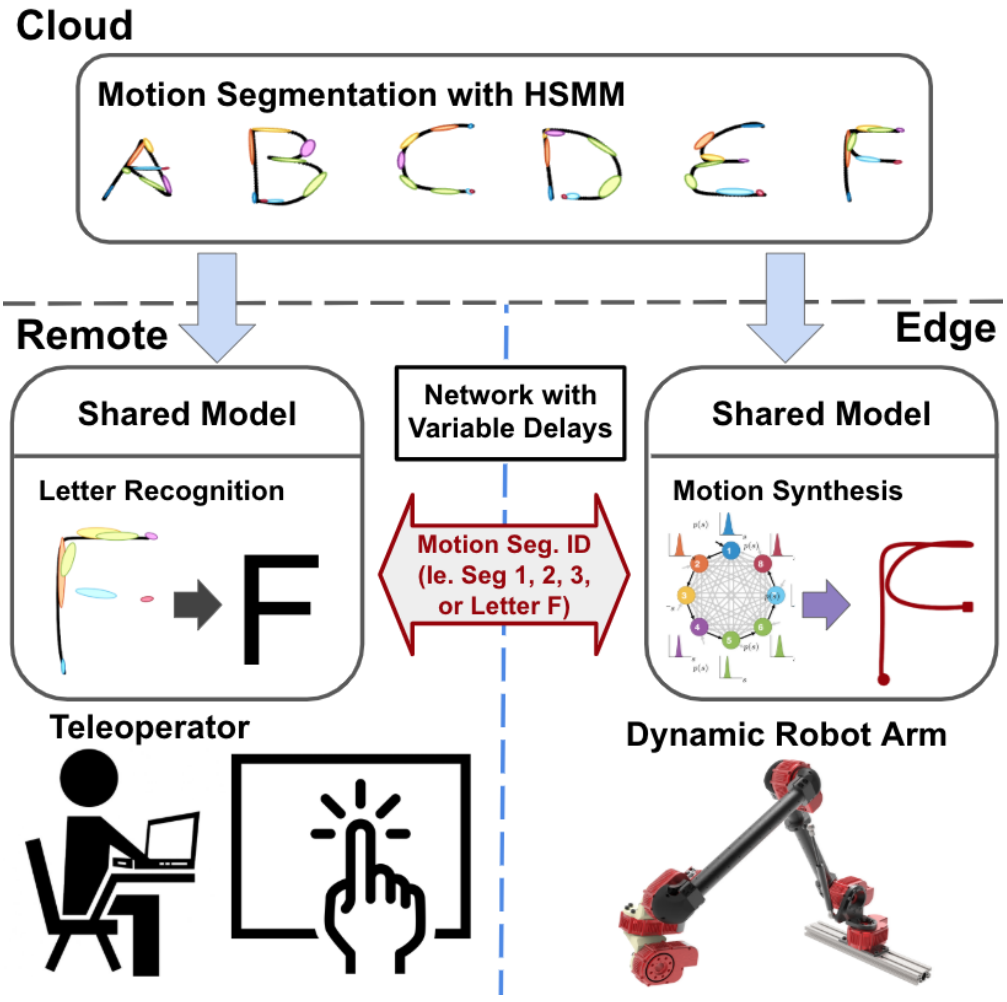

Figure 5.1: Intelligent Motion Segmentation and Synthesis System for Latency Mitigating: (Top) The Cloud encodes GMM/HSMM models for handwritten letters. (Left) The Remote tele-operator interface recognizes letters and motion segments based on user's partial demonstration, and send compact information to (Right) the Edge robotic controller where segments of motion are executed in a way that reduces effects of network latency.

at the Edge, so that the robot controller can catch up to the remote human demonstrations (Fig. 5.1 and Fig. 5.2).

Finally, we propose three different network latency mitigation protocols based on motion segment recognition and synthesis results. We implement the first two protocols to compare latency mitigation effects of the two protocols. The implementation of the third protocol is in progress. Our code is open-source [1]. We hope to build similar, closed-loop, interactive Fog Robotic tele-operation systems in the future for general human motions.

5.2 Contribution

This paper makes three contributions:

1. Probabilistic learning-based motion segmentation using HSMM to encode hand-written letters in the Cloud.
2. Generate synthetic motions at the Edge for stable, dynamic robot control.
3. Propose three network latency mitigation protocols that can anticipate and generate motions interactively based on partial demonstrations from the Remote tele-operator interface.

5.3 Related Work

Cloud, Edge, and Fog Robotics Cloud Robotics, introduced by James Kuffner in 2010 [75], refers to any robot or automation system that relies on either data or code from a network to support its operation [65]. It can be used to provide powerful machine learning systems for distributed robots. Network costs in the form of privacy, security latency, bandwidth, and reliability present a challenge in Cloud Robotics [124]. Fog Robotics, a variant of Cloud Robotics, has been introduced [45] and built [124] recently to bring cloud computing resources closer to the robot to balance storage, compute and networking resources between the Cloud and the Edge. A closed-loop Cloud-Edge hybrid controller was also built to control a dynamic balancing robot [130].

Latency Mitigation is important as unpredictable network latency presents primary challenge in building a closed-loop interactive robotic controller over the network. Network controlled system (NCS) often encounters similar problems [146][143] [137], and the delays were dealt with using predictive control and delay compensator. Previous work on intention recognition showed that intend prediction can assist tele-operator to perform robotic manipulation task under various network conditions [121]. Further, network latency can hide

[1] Source code, datasets, and algorithms are available at https://github.com/ajaytanwani/generative_models_public.

within robot motion execution in Cloud Robotics [98]. Motion synthesis using a generative model [122] is needed to achieve latency mitigation for interactive tele-operations.

Motion Segmentation and Synthesis for robotics has been explored with dynamic motion primitives (DMP) [93] [92], recurrent neural netorks (RNNs) [9], stochastic optimal control [8], HSMM and LQT [121] for both 2D trajectories, 3D trajectories, and human skeleton movements. These are a form of Learning from Demonstration (LfD), which learns a policy, model, or mapping between state and actions, from examples demonstrated by a teacher [4]. We differentiate our work by focusing on the latency mitigation protocol aspect of tele-operation with shared autonomy using existing motion segmenation and synthesis algorithms.

Tele-operation with Shared Autonomy can range from direct control to fully autonomous system, with human supervisory control in the middle. The more autonomous the tele-operation system is, the more tolerant it is against network delays [113]. Generative models such as HMM and HSMM has been used to build assisted tele-operation with shared autonomy between human and robot [121]. Our system for motion segmentation and synthesis fall into the supervisory control arena of the tele-operation spectrum.

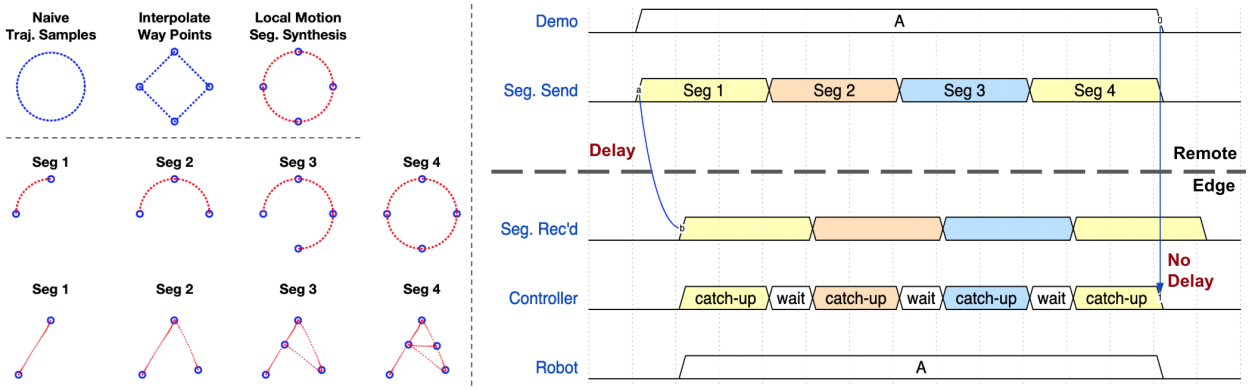

Figure 5.2: Motion Segmentation based Latency Mitigation Protocol. (I) Circle vs. Square This toy example shows the naive, undesired, and desired trajectories that can be generated for our system. (II) Stationary Point Motion Segmentation Here we show that we can perform motion segmentation by automatically detecting and grouping stationary way-points from data. We can then execute these motion segments in order to perform tele-operation in segments. (III) Latency Mitigation Protocol One: This illustrate our first latency mitigation protocol where segments of motion are transmitted to the Edge. The robot controller can in turn execute them in an elevated speed to eliminate network delays.

5.4 Problem Statement

Consider a teleoperator that controls a robot arm in a remote site. The teleoperator performs a partial trajectory $\boldsymbol{\xi}$ comprising of datapoints $\boldsymbol{\xi}_t \in \mathbb{R}^D$ at time t,

$$\boldsymbol{\xi} = \{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, ..., \boldsymbol{\xi}_t ..., \boldsymbol{\xi}_T\} \qquad t \in 1, 2...t \tag{5.1}$$

where $\boldsymbol{\xi}_t$ is a column vector of position, velocity, and acceleration, respectively, in 2D space, so $\boldsymbol{\xi}_t = [\vec{\boldsymbol{x}}_t, \dot{\vec{\boldsymbol{x}}}_t, \ddot{\vec{\boldsymbol{x}}}_t]^\top$.

We assume that the demonstration $\boldsymbol{\xi}$ comprises of the segments $\{z_i\}_{i=1}^D \in \mathbb{Z}$ to represent the demonstrated trajectory

$$\boldsymbol{\xi} = \{z_{T_1}^1, z_{T_2}^2, ...z_{T_D}^D\} \tag{5.2}$$

where $z_{T_D}^D$ is the D^{th} segment index with the duration of T_D. More precisely, each motion segment is

$$\boldsymbol{\xi}_{T_D}^D = \boldsymbol{\xi}_{t_D, t_D+T_D}^D \tag{5.3}$$

where t_D is the starting time of the segment, and $\boldsymbol{\xi}_{t_D}^D$ and $\boldsymbol{\xi}_{t_D+T_D}^D$ are the starting and ending point of the D^{th} segment. We define starting points $\boldsymbol{\xi}_{t_D}^D$ as the way-points of trajectory.

For sake of clarity, we assume that the trajectory demonstration corresponds to a hand-written letter l denoted as ${}^l\boldsymbol{\xi}$. In the first stage, the objective is to learn models of motion segments from teleoperator demonstrations. This is the encoding step. Subsequently during the decoding step, the learned segments are used for recognizing the intention of the teleoperator as writing a particular letter l from the partial demonstration sequence, and subsequently synthesize the motion for letter l on the remote robot. We denote the generated motion sequence on the robot with a hat as

$${}^l\hat{\boldsymbol{\xi}} = {}^l\hat{\boldsymbol{\xi}}_{T_1}^1, {}^l\hat{\boldsymbol{\xi}}_{T_2}^2, ... {}^l\hat{\boldsymbol{\xi}}_{T_D}^D \qquad l \in A, B, C, ..., Z \tag{5.4}$$

With the above definitions, we frame the motion segmentation and synthesis for the teleoperation task over the network:

1. **The Cloud**: learn models from data ${}^l\boldsymbol{\xi}$ to represent motion segments ${}^l\boldsymbol{\xi}_{T_D}^D$. Share the learned models with both the Remote and Edge controllers.

2. **The Remote**: Recognize motion segments ID D and letter ID l from partial human demonstrations $\boldsymbol{\xi}_{1,t}$, and send these high level command to the Edge.

3. **The Edge**: Given learned models, upon receiving D (segment ID), l (letter ID), and T (execution duration), synthesize motion segments ${}^l\hat{\boldsymbol{\xi}}^D$ or trajectory ${}^l\hat{\boldsymbol{\xi}}$ so that the robot can finish motion execution before the designated duration T.

5.5 Latency Mitigation Protocol I

Based on previous findings that robot motion executions can hide network latency [98], we propose latency mitigation protocols for tele-operation using an intelligent motion segmentation and synthesis system. Fig 5.2III presents the simplest form of this protocol. The Remote controller recognizes which segment the tele-operator is performing, and predicts where the intermediate target, or way-point of this segment is. The Remote controller sends motion segments to the Edge robot controller to execute, with a delay that includes both network latency and recognition delay. The Edge controller speeds up the motion execution so that the robot can catch up to the human demonstration segment-by-segment. In the end, the robot finishes the entire trajectory as if there were no delays in the network transmission.

We use a circle drawing example to illustrate why both motion segmentation and synthesis are needed for tele-operation. (Fig 5.2 I) In the naive case, drawing a circle requires the robot's end-effector to follow a densely sampled circle trajectory. If the samples were sent through network one-by-one, unpredictable variable delays could affect the circle drawing significantly.

If we break the circle into four segments, and only send out intermediate target points, the Edge controller would interpret the arcs as linear paths via interpolation, so that the robot would move in a square instead of a circle. Further, if both the Remote and the Edge share the shapes of these motion segments, the Edge can then fill in the gaps between way-points to reproduce motions similar to the tele-operator's. The shape information can either be raw motion segments pre-stored at the Edge, or learned generative models that can help the Edge synthesize motions.

5.6 Motion Segmentation with Stationary Point Heuristics

To establish a motion segmentation base-line with handwritten letter demonstrations, we first use well known minimum velocity and acceleration heuristics H [92] to automatically identify stationary points x_s.

$$x_s \in \{{}^l\boldsymbol{\xi}_t \mid H \approxeq 0\} \quad \text{where} \quad H = ||\dot{\vec{\boldsymbol{x}}}_t||^2 + ||\ddot{\vec{\boldsymbol{x}}}_t||^2 \tag{5.5}$$

We then perform K-means to group these stationary points into clusters $i \in K$ with centroid-means of μ^i. We then reassign cluster centroid IDs so that these IDs represent the sequential order in the demonstrations. Motion segments can then be defined as trajectories between adjacent clusters of stationary points.

$$\boldsymbol{\xi} = \{\boldsymbol{\xi}^1_{T_1}, \boldsymbol{\xi}^2_{T_2}, ...\boldsymbol{\xi}^K_{T_K}\} \text{ where } \mu^i \in \{\mu^1, \mu^2, ...\mu^K\} \tag{5.6}$$

We then share the re-ordered k-mean clusters and example trajectories to both the Remote and the Edge controllers, so that the Edge can replay pre-stored motion segments,

upon receiving the closest stationary point cluster recognized by the Remote based on below equations

$$i^{\text{ID}} := \underset{i \in \{1,\ldots,K\}}{\operatorname{argmin}} ||x_s - \mu^i||^2 \tag{5.7}$$

Control sequence re-played with Protocol One are shown in Fig. 5.2II for letter "A" and all letters in Fig. 5.7 I.

5.7 Probabilistic Motion Segmentation and Synthesis

With more advanced probabilistic generative models, not only can we perform motion segmentation, we can also generate synthetic motions. We encode and decode hand-written letter demonstrations with a HSMM in a probabilistic manner, and use LQT to synthesize motions. This technique is more general and appropriate for our application than the motion segmentation re-play base-line technique described in the last section.

5.7.1 Spatial Encoding/Decoding

We encode spatial information using Gaussian mixture model (GMM) so that each Gaussian mixture represent a motion segment in the trajectory. Given eight handwritten sample trajectories per letter represented by position and velocity $\boldsymbol{\xi}_t = [\vec{\boldsymbol{x}}_t; \dot{\vec{\boldsymbol{x}}}_t]$, we train a separate GMM model to encode each letter in the alphabet.

$$P(\boldsymbol{\xi}_t \mid \theta) = \sum_{i=1}^{K} \pi_i N(\boldsymbol{\xi}_t, \mid \mu_i, \Sigma_i) \tag{5.8}$$

where $P(\boldsymbol{\xi}_t \mid \theta)$ is the probability density function of sample point $\boldsymbol{\xi}_t$ conditioned on parameters $\theta = \{\pi_i, \mu_i, \Sigma_i\}_{i=1}^{K}$, a set of prior π_i, mean μ_i, and covariance matrix Σ_i for each of the K mixtures. The GMM are learned using Expectation-Maximization (EM) algorithms. The resulting GMM mixture models for each letter is shown in Fig. 5.3 I and Fig. 5.7 I.

During decoding, given a sample $\boldsymbol{\xi}_i$ and the GMM for a single letter, we decide the sample belong to which mixture $z_t = i$ using maximum log likelihood

$$i^{z_t} := \underset{i \in \{1,\ldots K\}}{\operatorname{argmax}} \quad log\big(\pi_i N(\boldsymbol{\xi}_t \mid \mu_i, \Sigma_i)\big) \tag{5.9}$$

We can this to decode which motion segment the current demonstration sample $\boldsymbol{\xi}_t$ belong to base on only spatial information.

5.7.2 Temporal Encoding/Decoding for Letter Recognition

In order to select which mixture component is required for motion generation, we need to recognize which letter the tele-operator is performing based on partial trajectory demonstrations. Therefore, we need to encode and decode both temporal and spatial information.

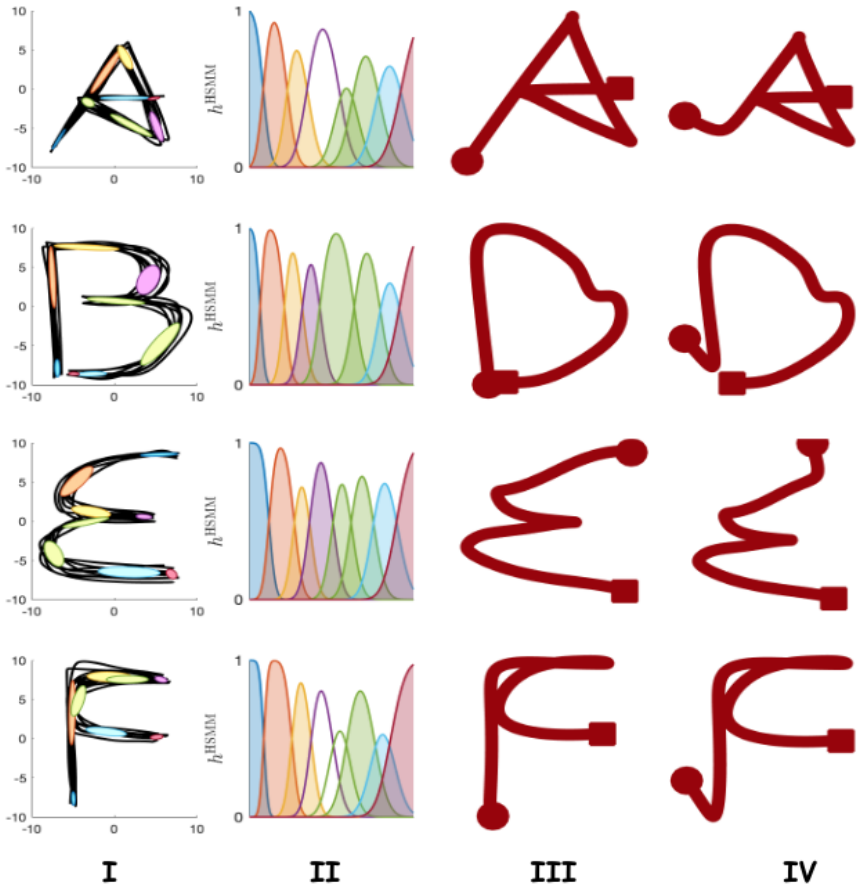

Figure 5.3: Trajectory generation with HSMM: (I) HSMM mixtures overlay on data We learn a HSMM for each letter from eight trajectory samples per letter. (II) HSMM State Probabilities of a given trajectory inferred through forward-backward Viberbi algorithm Generated Trajectories: (III) from the same start positions (circle) as the original demon, and (IV) from different start positions (circle) to show autonomy and robustness

We use both hidden semi-Markov model(HMM)(cite, Ajay) to encode and decode temporal state sequences.

Gaussian mixtures are used as latent states $z_t = i$ in HMM at time t. During encoding, the GMM-based HMM model parameterized by $\theta = \{\{a_{i,j}\}_{j=1}^{K}, \Pi_i, \mu_i, \Sigma_i\}_i$ learns: (a) transition probabilities $a_{i,j}$, (b) emission probabilities Π_i, (c) mean μ_i and covariance Σ_i via EM algorithm. Here, $a_{i,j}$ represents transition probabilities between the K Gaussians in GMM, and $i, j \in \{1, ...K\}$ are indexes of Gaussian mixtures.

We use the forward-backward Viterbi algorithm to decode the latent states from z_t from forward variable $\alpha = P(z_t = i, \boldsymbol{\xi}_1 ... \boldsymbol{\xi}_t \mid \theta)$. The probability of a data point $\boldsymbol{\xi}_t$ to be in state

i at time t given the partial observation $\{\boldsymbol{\xi}_1 ... \boldsymbol{\xi}_t\}$ can be calculated as:

$$h_{t,i} = P(z_t \mid \boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_t) = \frac{\alpha_{t,i}}{\sum_{k=1}^{K} \alpha_{t,k}} \tag{5.10}$$

where the forward variable α is

$$\alpha_{t,i} = \big(\sum_{j=1}^{K} \alpha_{t-1,i} a_{j,i}\big) N(\boldsymbol{\xi}_t \mid \mu_i, \Sigma_i) \tag{5.11}$$

HSMM generalized HMM by explicitly modeling an additional state duration probability so that state transition depends on not only current state, but also on the elapsed time in the current state. In HSMM, forward variable can be calculated:

$$\alpha_{t,i} = \sum_{s=1}^{min(s^{\max}, t-1)} \sum_{j=1}^{K} \alpha_{t-s,i} a_{j,i} N(s \mid \mu_i^s, \Sigma_i^s) \tag{5.12}$$

where s represent state duration steps in HSMM. For more details, please refer to [120] and [121].

To recognize the letter ID based on the available partial trajectory $\{\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_t\}$, we apply eq. (5.10) to all 26 HMMs with the parameters ${}^l\theta$ where $l \in \{A, B, ...Z\}$. The HMM model with the highest probabilities is selected as the letter that is being recognized based on partial trajectory:

$$l := \underset{l \in \{A,B,...Z\}}{\operatorname{argmax}} \; P(z_t \mid \boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_t; {}^l\theta) \tag{5.13}$$

5.7.3 Motion Synthesis based on Predicted State Sequence

To synthesize new motion, we first compute future state sequence $\boldsymbol{z_t}$ using the forward variable at time t.

$$\boldsymbol{z_t} = \{z_t, ..., z_T\} = \underset{i}{\operatorname{argmax}} \quad \alpha_{t,i} \tag{5.14}$$

We then build a reference trajectory distribution $N(\hat{\mu}_t, \hat{\Sigma}_t)$ by assigning the predicted parameters $\hat{\mu}_t$ and $\hat{\Sigma}_t$ at time t as the parameters μ_{z_t} and Σ_{z_t} for the predicted future states $\boldsymbol{z_t}$. Samples at time t can be generated from this reference trajectory distribution:

$$\hat{\boldsymbol{\xi}}_t \sim N(\hat{\mu}_t = \mu_{z_t}, \hat{\Sigma}_t = \Sigma_{z_t}) \quad \text{where} \quad t \in \{t...T\} \tag{5.15}$$

Finally, the Edge robot controller uses a linear quadratic tracking (LQT) to create a synthesized trajectory for a dynamical robot system:

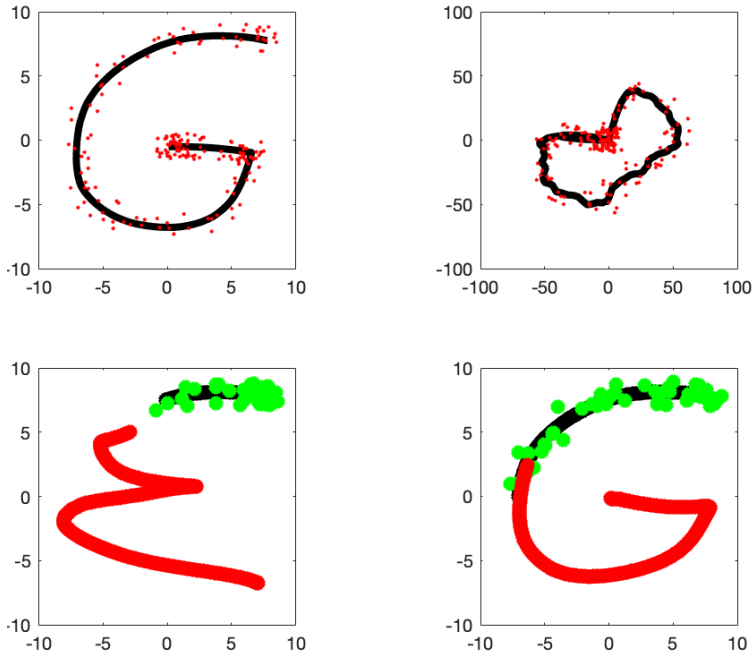

Figure 5.4: Interactive Recognition and Synthesis Trials: (Top) Uniformly Distributed Noise Injected to position (left, $\sigma = 2$ cm) and velocity (left, $\sigma = 20$ cm/sample) of trajectory "G" for benchmark trials. (Bottom) Synthesized Trajectory (red) based on letter recognitions of partial noisy demonstrations with different length (black with green noise). Shorter demonstration (left) cause more recognition failures, in this case, recognize the trajectory as "E". False recognition cause higher synthesis error. Longer partial demonstrations (right) help reducing both recognition and synthesis errors during tele-operation. See demo video https://youtu.be/fjlx5kXiMhc

$$c_t(\boldsymbol{\xi}_t, u_t) = \sum_{t=1}^{T} (\boldsymbol{\xi}_t - \hat{\mu}_t)^\top Q (\boldsymbol{\xi}_t - \hat{\mu}_t) + u_t^\top R_t u_t \tag{5.16}$$
$$s.t. \quad \dot{\boldsymbol{\xi}}_t = \boldsymbol{A}\boldsymbol{\xi}_t + \boldsymbol{B}u_t$$

for more details on LQT, refer to [120] and [121].

5.8 Modified Latency Mitigation Protocol II

Benchmarks of letter recognition and motion synthesis suggest that the motion recognition at the Remote controller is not reliable initially, and could lead to large synthesis error (see section VIII and Fig. 5.5I & II). We cannot expect to start latency mitigation from the very

beginning when the system is has low confidence of its prediction. Therefore, we modify Protocol One into Protocol Two to make it more practical for real-life implementations.

In this new protocol (Fig. 5.5 III), during the initial period when the Remote controller is not sure about which letter the tele-operator is demonstrating, the Edge controller follows the exact trajectory of the tele-operator, with a delay of course. As soon as the Remote controller recognizes and decides which letter is being draw, the Edge controller receives the letter ID, and commits to drawing the recognized letter through motion synthesis. This way, during the latency mitigation second phase, the Edge can catch up or surpass human demonstration, so that it can reduce or eliminate network latencies.

A further modification of this protocol, we denote as Protocol Three, is also possible. Illustrated in Fig. 5.6 left, the Edge controller in Protocol Three would synthesize and execute motion in segments instead of finishing the entire motion all at once during the second phase when the correct letter is recognized. Protocol Three can be more general, but is not implemented in this work.

5.9 Experiments and Results

We use a handwritten letter dataset to train the HSMM generative model. The dataset contains eight sample trajectories per letter in the alphabet. Each sample trajectory contains 200 sample points, each include 2D position in the range of -10 to 10 *cm*. By performing differentiation and interpolation on the positions at neighboring points, we extract additional velocity and acceleration features that are needed to encode motion segmentation and synthesis models. The same pre-processing is done on partial trajectories in decoding during human demonstrations. During encoding, we successfully learn 26 HSMM models (one for each letter), extract the state sequences, regenerate trajectories that have either the same or different starting points as the original trajectory (Fig. 5.3).

5.9.1 Recognition vs. Synthesis Error

There are two different kinds of inferencing during decoding: (1) recognition of letter given partial trajectory for HSMM model selection; (2) prediction of future state sequences so that a trajectory can be generated using LQT. Each associates with the two phases in the modified latency mitigation protocol–recognition and synthesis phase.

To quantitatively evaluate our system, we benchmark recognition and synthesis performance on variable length trajectories with injected noise (Fig. 5.4). Given a partial trajectory $\boldsymbol{\xi}_1, t$ ending at time t, recognition error is defined as the number of wrong letter recognition trials over total trials, whereas synthesis error is the average position L_2 error between the generated trajectory $\hat{\boldsymbol{\xi}}_{t,T}$ and the respective demonstration segment $\boldsymbol{\xi}_{t,T}$ from time t to finish time T.

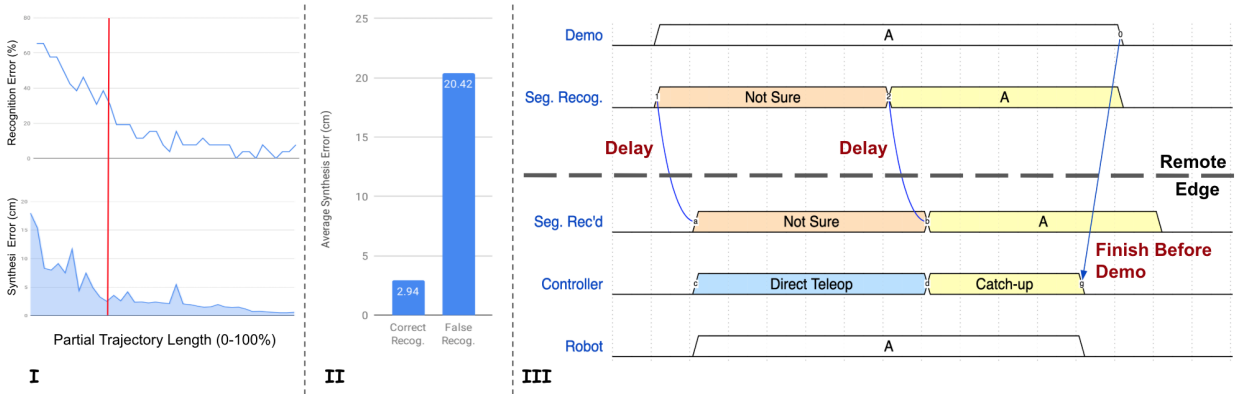

Figure 5.5: Recognize and Finish Latency Mitigation Protocol II: (I) Recognition (top) and Synthesis (bottom) Errors vs. Length of Trajectory shows that both errors reduce dramatically as demonstration progresses passing the 30% (red line) (II) Synthesis Error is much lower when recognition is correct, suggesting that recognition error is the main contributor to synthesis error. (III) Modified Latency Mitigation Protocol Two for handwritten letter segmentation and regeneration, based on findings in (I) and (II). See video for a demonstration when the Edge controller finish executing the synthetic motion before the Remote tele-operator https://youtu.be/fjlx5kXiMhc

$$\text{Recognition Error} = 1 - \frac{N_{Success}}{N_{Total}} \tag{5.17}$$

$$\text{Synthesis Error} = \frac{||\hat{\boldsymbol{\xi}}_{t,T} - \boldsymbol{\xi}_{t,T}||_2}{T - t} \tag{5.18}$$

Note that the synthesis error is normalized by the number of time samples generated, which accounts only part of the entire trajectory and has $T-t$ sample points. This way, L_2 distance of each sample contribute equally to synthesis error, so that we can compare synthesis error across partial trajectory generations with different lengths.

We conducted 10 trials per letter on partial trajectories with variable length $(0-100\%)$ injected with uniformly distributed random noise (variance $\sigma_{pos} = 2$ cm, $\sigma_{vel} = 2$ cm/sample, Fig. 5.4 top). Fig. 5.4 (bottom) shows examples of generated trajectory based on correct and wrong recognition results. In Fig. 5.5I, we plot both recognition and synthesis error of all trials against the length of the partial trajectory shown to the system.

We observe that both recognition and synthesis drop dramatically around 30% trajectory demonstration length. This suggests that recognition is not reliable for short trajectories with 30% length, and it becomes more reliable as the demonstration progresses (Fig. 5.4 bottom).

It also suggests a strong correlation between recognition and synthesis error, as, naturally, synthesis error would grow dramatically if the letter recognition is wrong. We show that recognition error contributes to the majority of the synthesis error in Fig. 5.5 II where the synthesis errors of all the trials with correct and wrong recognitions are compared against each other.

5.9.2 Latency Mitigation Effects

We want to observe how much latency the system can tolerate for the two latency protocols. Protocol One with stationary point segmentation is used as base-line. Intuitively, Protocol One can tolerate delays at most to a fraction of the length of segments. The length of the four segments of the letter "A" are 63, 43, 81, 12 sample points. Assuming that the robot can move twice as fast as the demonstrator, then the system can tolerate up to 31, 21, 40, and 6 sample points. Therefore it can mitigate up to 0.5, 0.3, 0.7, and 0.1 seconds of delays respectively when a 60 Hz sample rate is assumed. Any delay that is lower than these durations, unpredictable it might be, is going to be eliminated.

Protocol Two can tolerate even more delays as motion synthesis allow it to be autonomous over the entire second phase of the protocol, after correct letter recognition. In the demo video for Protocol Two (https://youtu.be/fjlx5kXiMhc), we show the interaction between the Remote demonstrator and the Edge controller when drawing letter "G", "H", "B", "P", and "K". The synthesized trajectory is red during first phase, and it changes when the system is not sure which letter it is early on in the demonstration. After recognize the letter with high confidence, the Remote controller execute the motion at 2x speed, so that the robot can finish the motion even before the tele-operator.

The second phase lasts 153, 107, 83, 61, and 127 samples for letters "G", "H", "B", "P", and "K", which lasts 2.6, 2.8, 1.4, 1.0, and 2.1 seconds. Half of that period, or 1.3, 1.4, 0.7, 0.5, and 1.0 seconds, can be used to mitigate latency. Protocol Two has more tolerance against unpredictable latency than Protocol One, because it has longer autonomous motion.

To gain high confidence in letter recognition, we do use a 40 sample window (0.6 seconds) during which all the recognition results have to be the same in order to enter the second phase. Therefore, additional recognition delay is introduced to trade for the price to eliminate network delays during the second phase. We believe that the benefit of eliminating not only unpredictable network delays, but also potential instabilities in a dynamical system justified paying the price of recognition delay.

5.10 Discussion and Future Work

We present an intelligent latency mitigation tele-operation system for handwritten letter drawing. Motion segmentation and synthesis are used to reduce the effects of network latency by hiding network delays inside generated synthetic motion segments. We use two different algorithms to perform motion segmentaion based on either (1) stationary points heuristics

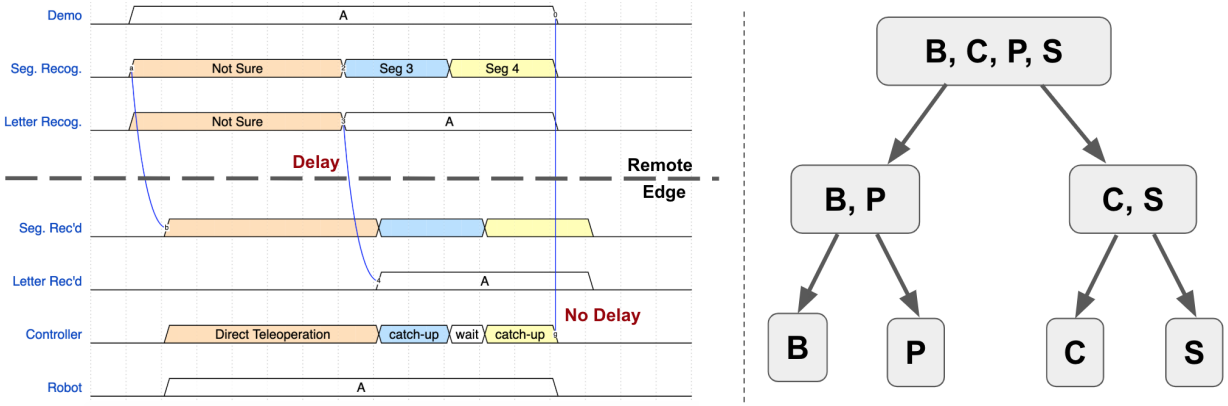

Figure 5.6: Possible Extension: A Behavior-based Hierarchical Latency Mitigation Protocol III: (Left) Protocol Three that recognize both letters and motion segments (Right) Proposing a future hierarchical GMM/HSMM that can recognize and generate longer motion segments for the entire alphabet.

with K-means or (2) HSMM state sequences. The HSMM method is desirable as it can also generate motion segments for motion synthesis. We further introduce and evaluate latency mitigation communication protocols based on recognition and synthesis error benchmarked under this intelligent system.

There are trade-offs, however, in this latency mitigation system. Although we reduce the effect of unpredictable network latency on a dynamical system, we introduce recognition delays into the system that further reduce the time period for robot controller to catch up to the tele-operator. Therefore, longer motion segments are more desirable, because they leave more room for the Remote to recognize the segment and for the Edge robot controller to catch up. Under the current HSMM system, length of the segments are defined by state sequence of Gaussian mixture. To maximize motion segmentation length, we should reduce the number of Gaussian mixtures. However, there is a limit in doing so, because fewer Gaussian mixtures may not be able to fully characterize the handwritten motion, therefore, affect both recognition and synthesis error.

One way to maximize the motion segment length while keeping sufficient Gaussian mixtures for representation is to group neighboring Gaussian mixture states into a single segment. This would require a new HSMM that recognize groups of GMM states based on the recognition of the first GMM states in the mega-group, and then generates mega-segment. We would need to hierarchically group Gaussian mixtures to represent mega-segments of the entire dataset, so that a hierarchical HSMM can be recognized and regenerated mega-segments based on a execution tree (Fig. 5.6 Right).

For example, in the letter set $\{B, C, P, S\}$, super-nodes $\{B, P\}$ and $\{C, S\}$ contain letters

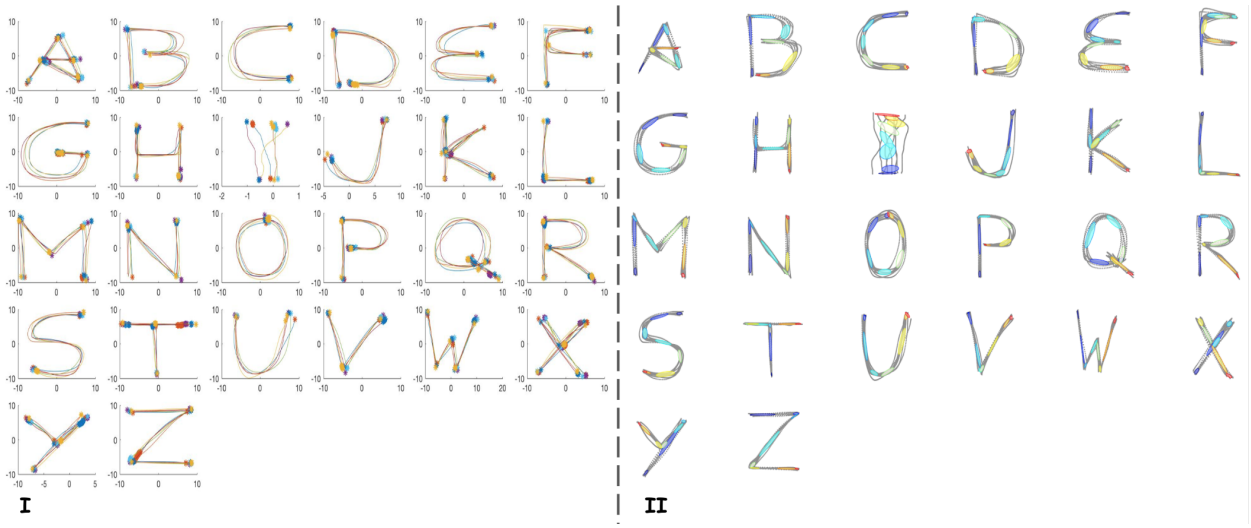

Figure 5.7: All Handwritten Letter Motion Segmentation using (I) Stationary Point Heuristics and (II) Gaussian Mixture Models Clusters

that are similar in the drawing process. In such hierarchical HSMM model, when drawing the letter B, the model should traverse top-to-bottom in the tree to command the Edge controller to execute motion segment P first, the meta-segment shared by B and P. The controller should then decide whether to finish drawing letter B with an additional motion segment or not at super-node $\{B, P\}$, upon additional demonstration given by the tele-operator. This hierarchical HSMM model is still work-in-progress. See our demo video for intuition of hierarchical structures when drawing letter B and P.

Finally, our system is limited to tele-operation for 2D handwritten letter drawing, as specific set of motions. HSMM is more general, and can encode and decode 3D motion trajectories, both in end-effector space and joint space. [120] We plan to extend our work into segment and reproduce human skeleton motions in 3D for interactive human-robot imitation and interactions.

Notes

This chapter has appeared in a peer-reviewed conference paper:

Nan Tian, Ajay Kumar Tanwani, Ken Goldberg and Somayeh Sojoudi, *Mitigating Network Latency in Cloud-Based Teleoperation using Motion Segmentation and Synthesis.* ICRA 2019.

Chapter 6

Discussion and Conclusion

6.1 Overview

In this thesis, we built a 'hybrid' Cloud-Edge Robotic system for pHRI applicaitons in progressive steps. A final architecture for both teleoperation and proximate interactions 1.2. We explored how to use this system to support various robotic tasks, ranging from graspings and manipulations to humanoid gesture based imitations, from teleoperations to proximate interactions, from quansi-static to highly dynamic robotic tasks. We identified that unpredictable network latencies were the main drawback of using such system on a global scale. We then leveraged robotic learning to perform motion segmentation and synthesis to mitigate such unpredictable network latencies in teleoperation.

6.2 Fluent Physical Human Robot Interactions

The key motivation of developing the 'hybrid' system under a joint Cloud-Edge controller is to support fluent physical human robot interactions using Cloud Robotic services that are powerful, modular, distributed, but heterogeneous in nature. The unpredictability of network delays are caused by the heterogeneous computing environment in Cloud services. If not managed, such delays can cause safety, stability, and controllability issues in dynamic robotic control [38]. Chapter 5 of this thesis proposed one possible system to mitigate such network latencies while controlling a dynamic robotic system to assist human teleoperation. It learns human motion segments through data, and synthesize similar motion segments on the robot upon receiving of discrete behavior based command. The robot execute the synthesized motion faster than the demonstrator so that the network latencies can be mitigated.

This idea leverages the human motion dichotomy which suggests that all human motions can be categorized into mean-based motions and functional motions [3]. Mean-based motions convey a meaning whereas functional motions carry out tasks such as pickings and graspings. We have implemented both class of motion–semaphore in Chapter 4 is an example of mean-based motions, whereas objects pickups in Chapter 2 and 3 perform functional motions.

Under the framework of behavior-based robotics [5], we can extract a behavior graph from mean-based motions and use this graph to organize and command functional motion segments. Our work in Chapter 5 were limited to recognizing a single concept to drive multiple dynamic motion primitives, primarily because the robotic task we selected, handwritten letter, were relatively simple. Although such simplicity helped us to develop components of network mitigation protocols, a more sophisticated robotic tasks, such as human gesture imitation [100], would help guiding future researches to build a general behavior-based 'hybrid' architecture. Such a system can organize motion primitives into behavior map automatically, so that the Cloud can command complex functional motions using a behavior graph. As one suggested in Figure 5.6, such hybrid system that can recognize multiple consecutive behaviors and synthesize multiple motion segments can be build in the future.

At the Edge level, functional motion segments generated at the low level controller served as fundamental building blocks to control dynamic robotic arms. They defined different control laws to control robot arms that can be switched from one to another. Different control framework can be used to synthesize functional motion segments, such as the GMM/HSMM/LQT task parameter space formulation [121] we used and dynamic motion primitives [118] others have used. Compliance and stability analysis are important aspects of pHRIs and hybrid systems [38], and they should be done at the level of motion generators in the future.

6.3 Recognition Delays and Delay Tolerance

Although we could mitigate network latencies through motion segmentation and synthesis, we actually introduced additional recognition delays using the shared autonomy system shown in 4.1. The recognition delays are hard to benchmark, yet, it imposes additional timing costs associated with our Cloud-Edge hybrid system. In fact, recognition delays can be significantly higher than even the global network delays. As shown in Figure 5.6, recognition delay can be up to 1-2 seconds, or about 30% of the entire letter demonstration, whereas the worst global network delays with video streams are only in the range of 0.5 second. Fortunately, our system can tolerate every long delays because it can hide delays within the motion execution. However, there is a limit were such mitigation fails, about 50% of the motion segment execution, as the robot arm needs time to catchup to the demonstrators before the end of each demonstrated motion segment.

As recognition delay dominates network delays, finding ways to minimize recognition delays is an important aspect for future investigations. More contemporary motion recognition frameworks using recurrent neural networks [29], variational auto-encoders [99], deep reinforcement learning [81, 82], and meta-learning [33] may help to reduce recognition delays, though they require significantly more data during training compared to GMM/HSMM models. Further investigation on human gesture datasets with labeled segments is needed to minimize and benchmark recognition delays for the Cloud-Edge hybrid system.

6.4 Perception in the Cloud or at the Edge

We have advocated hosting perception servers in the Cloud in this thesis. The advantages to use the Cloud is modularity, maintainability, computational power, centralized controls, and shared data and perception streams. Recent developments on neural network compression [47, 86] and low power real-time perception systems [46, 145]seems to contradict the idea that Cloud is the go-to-place to host complex perception services, as ARM based Edge controllers could have significant computational power for a compressed neural network to run in real-time [59].

We, however, believe that powerful neural network based model deployed at local edge robots complements Cloud Robotics, especially within the hybrid Cloud-Edge architecture. First, a good architecture should be capable of functionalities that can accommodate full range of applications. A Cloud-Edge architecture accommodates both intelligent perceptions and dynamic robotic controls, as shown in 3.2. Second, even as the Edge controller becomes more powerful, a centralized cloud computing will always be more powerful and more efficient than local Edge controllers to provide certain functionalities, for example, neural network trainings, collect and host large amount of shared data, centralized and modularized control, shared global perception, and manage learned models to distributed agents. Finally, a Cloud Robotic framework actually contains the Cloud and the Edge controllers as well as the Internet connections in between. Not only does it provide access to the Cloud computation, it also provides means to long range communications that is critical for teleoperation applications.

Improvement of the Edge controller can further enrich the capability of Cloud Robotics, as more capable Cloud-Edge hybrid controllers can be built. For example, if a local neural network on the robot can be used to process video streams into compact behavior representations, such as semaphore, the Cloud controller can then use these representations to control high level robotic behavior using a global graph neural network hosted in the Cloud. Finally, the Cloud controller can perceive and learn from global representations collected from all the robots without having to stream videos from the robots, hence, having a more powerful and capable Edge controller will further increase the capacity and scalability of Cloud Robotics. We look forward to a world filling with service robots that can make our life easier, supported by Cloud Robotics.

Bibliography

[1] *ABB YuMi Datasheet*. ABB Group. 2015.

[2] Waleed Abdulla. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. https://github.com/matterport/Mask_RCNN. 2017.

[3] Baris Akgun et al. "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective". In: *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*. 2012, pp. 391–398.

[4] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[5] R Arkin. *Behavior-Based Robotics/Cambridge, MA*. 1998.

[6] Rajesh Arumugam et al. "DAvinCi: A cloud computing framework for service robots". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 3084–3089.

[7] Kostas Bekris et al. "Cloud automation: Precomputing roadmaps for flexible manipulation". In: *IEEE Robotics & Automation Magazine* 22.2 (2015), pp. 41–50.

[8] Daniel Berio, Sylvain Calinon, and Frederic Fol Leymarie. "Dynamic Graffiti Stylisation with Stochastic Optimal Control". In: *Proceedings of the 4th International Conference on Movement Computing*. ACM. 2017, p. 18.

[9] Daniel Berio et al. "Calligraphic stylisation learning with a physiologically plausible model of movement and recurrent neural networks". In: *Proceedings of the 4th International Conference on Movement Computing*. ACM. 2017, p. 25.

[10] Paul J Besl and Neil D McKay. "Method for registration of 3-D shapes". In: *Sensor Fusion IV: Control Paradigms and Data Structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–607.

[11] Antonio Bicchi and Vijay Kumar. "Robotic grasping and contact: A review". In: *ICRA*. Citeseer. 2000, pp. 348–353.

[12] A Birk et al. "Dexterous Underwater Manipulation from Distant Onshore Locations". In: *IEEE Robotics and Automation Magazine* ().

[13] Kanad K Biswas and Saurav Kumar Basu. "Gesture recognition using microsoft kinect®". In: *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*. IEEE. 2011, pp. 100–103.

[14] Jeannette Bohg et al. "Data-driven grasp synthesis—a survey". In: *IEEE Transactions on Robotics* 30.2 (2014), pp. 289–309.

[15] Flavio Bonomi et al. "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, pp. 13–16.

[16] Cynthia Breazeal. "Social interactions in HRI: the robot view". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 34.2 (2004), pp. 181–186.

[17] Cynthia Breazeal and Brian Scassellati. "Robots that imitate humans". In: *Trends in cognitive sciences* 6.11 (2002), pp. 481–487.

[18] GABRIEL Brown. "Ultra-reliable low-latency 5G for industrial automation". In: *Heavy Reading white paper for Qualcomm* (2018).

[19] Sylvain Calinon. "A tutorial on task-parameterized movement learning and retrieval". In: *Intelligent Service Robotics* 9.1 (2016), pp. 1–29.

[20] Sylvain Calinon, Florent Guenter, and Aude Billard. "Goal-directed imitation in a humanoid robot". In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 299–304.

[21] Sylvain Calinon, Florent Guenter, and Aude Billard. "On learning, representing, and generalizing a task in a humanoid robot". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37.2 (2007), pp. 286–298.

[22] Zhe Cao et al. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields". In: *arXiv preprint arXiv:1812.08008*. 2018.

[23] Zhe Cao et al. "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *CVPR*. 2017.

[24] CatClifford. *Look inside the hospital in China where coronavirus patients were treated by robots*. 2020. URL: https://www.cnbc.com/2020/03/23/video-hospital-in-china-where-covid-19-patients-treated-by-robots.html.

[25] François Chaumette and Seth Hutchinson. "Visual servo control. I. Basic approaches". In: *IEEE Robotics & Automation Magazine* 13.4 (2006), pp. 82–90.

[26] Sandeep Chinchali et al. "Network Offloading Policies for Cloud Robotics: a Learning-based Approach". In: *arXiv preprint arXiv:1902.05703* (2019).

[27] Hao Dang and Peter K Allen. "Learning grasp stability". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 2392–2397.

[28] Myron A Diftler et al. "Robonaut 2-the first humanoid robot in space". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2178–2183.

[29] Yong Du, Wei Wang, and Liang Wang. "Hierarchical recurrent neural network for skeleton based action recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1110–1118.

[30] Gabriele Ermacora, Stefano Rosa, and Antonio Toma. "Fly4SmartCity: A cloud robotics service for smart city applications". In: *Journal of Ambient Intelligence and Smart Environments* 8.3 (2016), pp. 347–358.

[31] C. Ferrari and J. Canny. "Planning Optimal Grasps". In: *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)'92*. 1992.

[32] Carlo Ferrari and John Canny. "Planning optimal grasps". In: *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. IEEE. 1992, pp. 2290–2295.

[33] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1126–1135.

[34] Chelsea Finn and Sergey Levine. "Deep visual foresight for planning robot motion". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 2786–2793.

[35] Terrence Fong, Illah Nourbakhsh, and Kerstin Dautenhahn. "A survey of socially interactive robots". In: *Robotics and autonomous systems* 42.3-4 (2003), pp. 143–166.

[36] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[37] Georgia Gkioxari, Ross Girshick, and Jitendra Malik. "Contextual action recognition with r* cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1080–1088.

[38] Faik Goktas, Jonathan M Smith, and R Bajcsy. "Telerobotics over communication networks". In: *Proceedings of the 36th IEEE Conference on Decision and Control*. Vol. 3. IEEE. 1997, pp. 2399–2404.

[39] K. Goldberg and R. Siegwart. *Beyond Webcams: An Introduction to Online Robots*. MIT Press, 2002.

[40] Ken Goldberg. "Robots and the return to collaborative intelligence". In: *Nature Machine Intelligence* 1.1 (2019), pp. 2–4.

[41] Ken Goldberg. *The Robot in the Garden: Telerobotics and Telepistemology in the Age of the Internet*. Mit Press, 2001.

[42] Corey Goldfeder and Peter K Allen. "Data-driven grasping". In: *Autonomous Robots* 31.1 (2011), pp. 1–20.

[43] Michael A Goodrich and Alan C Schultz. "Human-robot interaction: a survey". In: *Foundations and trends in human-computer interaction* 1.3 (2007), pp. 203–275.

[44] Bruno Duarte Gouveia et al. "Computation sharing in distributed robotic systems: A case study on SLAM". In: *IEEE Transactions on Automation Science and Engineering* 12.2 (2015), pp. 410–422.

[45] SLK Chand Gudi et al. "Fog robotics: An introduction". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2017.

[46] Kaiyuan Guo et al. "From model to FPGA: Software-hardware co-design for efficient neural network acceleration". In: *2016 IEEE Hot Chips 28 Symposium (HCS)*. IEEE. 2016, pp. 1–27.

[47] Song Han, Huizi Mao, and William J Dally. "A deep neural network compression pipeline: Pruning, quantization, huffman encoding". In: *arXiv preprint arXiv:1510.00149* 10 (2015).

[48] Ioannis Havoutis and Sylvain Calinon. "Learning assistive teleoperation behaviors from demonstration". In: *Safety, Security, and Rescue Robotics (SSRR), 2016 IEEE International Symposium on*. IEEE. 2016, pp. 258–263.

[49] Andrew J. Hawkins. *Nuro is using delivery robots to help health care workers fighting COVID-19*. 2020. URL: https://www.theverge.com/2020/4/22/21231466/nuro-delivery-robot-health-care-workers-food-supplies-california.

[50] Kaiming He et al. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[51] *HEBI Robotics*. URL: https://www.hebirobotics.com/.

[52] Masato Hirose and Kenichi Ogawa. "Honda humanoid robots development". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 365.1850 (2007), pp. 11–19.

[53] Seth Hutchinson, Gregory D Hager, and Peter I Corke. "A tutorial on visual servo control". In: *IEEE transactions on robotics and automation* 12.5 (1996), pp. 651–670.

[54] *IEEE Networked Robots Technical Committee*. URL: http://www-users.cs.umn.edu/~isler/tc/.

[55] F Iida, M Tabata, and F Hara. "Generating personality character in a Face Robot through interaction with human". In: *7th IEEE International Workshop on Robot and Human Communication*. 1998, pp. 481–486.

[56] M. Inaba. "Remote-brained robots". In: *Proc. International Joint Conference on Artificial Intelligence*. 1997, pp. 1593–1606.

[57] Jan Prins Jeffrey Ichnowski and Ron Alterovitz. "Cloud based Motion Plan Computation for Power Constrained Robots". In: *2016 Workshop on the Algorithmic Foundations of Robotics*. WAFR. 2016.

[58] Edward Johns, Stefan Leutenegger, and Andrew J Davison. "Deep Learning a Grasp Function for Grasping under Gripper Pose Uncertainty". In: *arXiv preprint arXiv:1608.02239* (2016).

[59] Sunggoo Jung et al. "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2539–2544.

[60] Angjoo Kanazawa et al. "End-to-end Recovery of Human Shape and Pose". In: *Computer Vision and Pattern Regognition (CVPR)*. 2018.

[61] Daniel Kappler, Jeannette Bohg, and Stefan Schaal. "Leveraging big data for grasp planning". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 4304–4311.

[62] Roger P Karrer, Antonio Pescapé, and Thomas Huehn. "Challenges in second-generation wireless mesh networks". In: *EURASIP Journal on Wireless Communications and Networking* 2008 (2008), pp. 1–10.

[63] Ajay Kattepur, Hemant Kumar Rath, and Anantha Simha. "A-priori estimation of computation times in fog networked robotics". In: *2017 IEEE International Conference on Edge Computing (EDGE)*. IEEE. 2017, pp. 9–16.

[64] B. Kehoe. "Cloud-based Methods and Architectures for Robot Grasping". PhD thesis. Univ. of California, Berkeley, 2014.

[65] Ben Kehoe et al. "A survey of research on cloud robotics and automation". In: *IEEE Transactions on Automation Science and Engineering* 12.2 (2015), pp. 398–409.

[66] Ben Kehoe et al. "Cloud-based robot grasping with the google object recognition engine". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 4263–4270.

[67] Ben Kehoe et al. "Cloud-based robot grasping with the google object recognition engine". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 4263–4270.

[68] J. Kim et al. "Physically based grasp quality evaluation under uncertainty". In: *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)'12*. 2012.

[69] Junggon Kim et al. "Physically based grasp quality evaluation under pose uncertainty". In: *IEEE Transactions on Robotics* 29.6 (2013), pp. 1424–1439.

[70] Dan Krauth and Wabc. *Coronavirus: NY nursing home deploys robot to combat COVID-19*. 2020. URL: https://abc7ny.com/health/ny-nursing-home-deploys-robot-to-combat-covid-19/6198936/.

[71] Sanjay Krishnan et al. "ActiveClean: interactive data cleaning for statistical modeling". In: *Proceedings of the VLDB Endowment* 9.12 (2016), pp. 948–959.

[72] Sanjay Krishnan et al. "Privateclean: Data cleaning and differential privacy". In: *Proceedings of the 2016 International Conference on Management of Data*. ACM. 2016, pp. 937–951.

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

[74] John Kubiatowicz et al. "Secure Fog Robotics Using the Global Data Plane". In: *NSF Proposal* (2018).

[75] James J Kuffner et al. "Cloud-enabled robots". In: *IEEE-RAS international conference on humanoid robotics, Nashville, TN*. 2010.

[76] Scott Kuindersma et al. "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot". In: *Autonomous Robots* 40.3 (2016), pp. 429–455.

[77] Michael Laskey et al. "Dart: Noise injection for robust imitation learning". In: *arXiv preprint arXiv:1703.09327* (2017).

[78] Michael Laskey et al. "Multi-armed bandit models for 2D grasp planning with uncertainty". In: *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2015, pp. 572–579.

[79] Michael Laskey et al. "Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations". In: *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2016, pp. 827–834.

[80] Alex X Lee, Sergey Levine, and Pieter Abbeel. "Learning visual servoing with deep features and fitted q-iteration". In: *arXiv preprint arXiv:1703.11000* (2017).

[81] Sergey Levine and Pieter Abbeel. "Learning neural network policies with guided policy search under unknown dynamics". In: *Advances in Neural Information Processing Systems*. 2014, pp. 1071–1079.

[82] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *arXiv preprint arXiv:1603.02199* (2016).

[83] Pusong Li et al. "Dex-Net as a Service (DNaaS): A Cloud-Based Robust Robot Grasp Planning System". In: ().

[84] Rui Li et al. "Localization and manipulation of small parts using gelsight tactile sensing". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 3988–3993.

[85] Changliu Liu and Masayoshi Tomizuka. "Robot safe interaction system for intelligent industrial co-robots". In: *arXiv preprint arXiv:1808.03983* (2018).

[86] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. "Thinet: A filter level pruning method for deep neural network compression". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5058–5066.

[87] John Lygeros, Claire Tomlin, and Shankar Sastry. "Hybrid systems: modeling, analysis and control". In: *preprint* (1999).

[88] Jeffrey Mahler et al. "Dex-Net 1.0: A Cloud-Based Network of 3D Objects for Robust Grasp Planning Using a Multi-Armed Bandit Model with Correlated Rewards". In: *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)*. 2016.

[89] Jeffrey Mahler et al. "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics". In: *arXiv preprint arXiv:1703.09312* (2017).

[90] Alessandro Manzi et al. "Design of a cloud robotic system to support senior citizens: the KuBo experience". In: *Autonomous Robots* (2016), pp. 1–11.

[91] Gerard T McKee and Paul S Schenker. "Networked robotics". In: *Sensor Fusion and Decentralized Control in Robotic Systems III*. Vol. 4196. International Society for Optics and Photonics. 2000, pp. 197–209.

[92] Franziska Meier, Evangelos Theodorou, and Stefan Schaal. "Movement segmentation and recognition for imitation learning". In: *Artificial Intelligence and Statistics*. 2012, pp. 761–769.

[93] Franziska Meier et al. "Movement segmentation using a primitive library". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 3407–3412.

[94] Giorgio Metta et al. "The iCub humanoid robot: an open platform for research in embodied cognition". In: *Proceedings of the 8th workshop on performance metrics for intelligent systems*. ACM. 2008, pp. 50–56.

[95] Gajamohan Mohanarajah et al. "Rapyuta: A cloud robotics platform". In: *IEEE Transactions on Automation Science and Engineering* 12.2 (2015), pp. 481–493.

[96] Carla Mouradian et al. "A comprehensive survey on fog computing: State-of-the-art and research challenges". In: *IEEE Communications Surveys & Tutorials* 20.1 (2017), pp. 416–464.

[97] Richard M Murray et al. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[98] Xinhe Ren Michael Yu Robert Zhang Bill Huang Ken Goldberg Nan Tian Benjamin Kuo and Somayeh Sojoudi. "A Cloud-Based Robust Semaphore Mirroring System for Social Robots". In: *arXiv preprint arXiv:1703.09327* (2017).

[99] Daehyung Park, Yuuna Hoshi, and Charles C Kemp. "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1544–1551.

[100] Xue Bin Peng et al. "Sfv: Reinforcement learning of physical skills from videos". In: *ACM Transactions on Graphics (TOG)* 37.6 (2018), pp. 1–14.

[101] Kun Qian, Jie Niu, and Hong Yang. "Developing a gesture based remote human-robot interaction system using Kinect". In: *International Journal of Smart Home* 7.4 (2013).

[102] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[103] International Federation of Robotics. *Executive Summary World Robotics 2017 Service Robots*. 2017 (accessed September 15, 2018). URL: https://ifr.org/downloads/press/Executive_Summary_WR_Service_Robots_2017.pdf.

[104] International Federation of Robotics. *Introduction into Service Robots*. 2016 (accessed September 15, 2018). URL: https://ifr.org/img/office/Service_Robots_2016_Chapter_1_2.pdf.

[105] SoftBank Robotics. "Nao www. ald. softbankrobotics. com/en/robots/nao". In: *Last accessed* 20 (2017).

[106] Softbank Robotics. "Pepper". In: *Softbank Robotics* (2016).

[107] *Robots replace Japanese students at graduation amid coronavirus*. 2020. URL: https://www.reuters.com/article/us-health-coronavirus-japan-remote-gradu/robots-replace-japanese-students-at-graduation-amid-coronavirus-idUSKBN21P0XI.

[108] *Robots Versus Wizards Chess Set*. URL: http://www.thingiverse.com/thing:351119.

[109] Ludovico Orlando Russo et al. "A Novel Cloud-Based Service Robotics Application to Data Center Environmental Monitoring". In: *Sensors* 16.8 (2016), p. 1255.

[110] Gildardo Sánchez and Jean-Claude Latombe. "A single-query bi-directional probabilistic roadmap planner with lazy collision checking". In: *Robotics Research*. Springer, 2003, pp. 403–417.

[111] Azad Shademan, Amir-Massoud Farahmand, and Martin Jägersand. "Robust jacobian estimation for uncalibrated visual servoing". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 5564–5569.

[112] A. Singh. "Benchmarks for Cloud Robotics". PhD thesis. Univ. of California, Berkeley, 2016.

[113] D. Song, A. K. Tanwani, and K. Goldberg. "Networked-, Cloud- and Fog-Robotics". In: *Robotics Goes MOOC*. Ed. by B. Siciliano. Springer, 2019.

[114] Dezhen Song, Kenneth Y Goldberg, and Nak Young Chong. *Networked Telerobots*. 2008.

[115] Dezhen Song et al. "Networked-, cloud-and fog-robotics". In: *Springer* (2019).

[116] Christopher Stanton, Anton Bogdanovych, and Edward Ratanasena. "Teleoperation of a humanoid robot using full-body motion capture, example movements, and machine learning". In: *Proc. Australasian Conference on Robotics and Automation*. 2012.

[117] Nick Statt. *Boston Dynamics' Spot robot is helping hospitals remotely treat coronavirus patients*. 2020. URL: https://www.theverge.com/2020/4/23/21231855/boston-dynamics-spot-robot-covid-19-coronavirus-telemedicine.

[118] Freek Stulp, Evangelos A Theodorou, and Stefan Schaal. "Reinforcement learning with sequences of motion primitives for robust manipulation". In: *IEEE Transactions on robotics* 28.6 (2012), pp. 1360–1370.

[119] Shenglong Tang et al. "Cloud Robotics: Insight and Outlook". In: *International Conference on Industrial IoT Technologies and Applications*. Springer. 2016, pp. 94–103.

[120] A. K. Tanwani. "Generative Models for Learning Robot Manipulation Skills from Humans". PhD thesis. Ecole Polytechnique Federale de Lausanne, Switzerland, 2018.

[121] A. K. Tanwani and S. Calinon. "A generative model for intention recognition and manipulation assistance in teleoperation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. 2017, pp. 43–50. DOI: 10.1109/IROS.2017.8202136.

[122] A. K. Tanwani et al. *Generalizing Robot Imitation Learning with Invariant Hidden Semi-Markov Models*. 2018. arXiv: 1811.07489 [cs.RO].

[123] Ajay Kumar Tanwani et al. "A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 4559–4566.

[124] Ajay Kumar Tanwani et al. "A Fog Robotics Approach to Deep Robot Learning: Application to Object Recognition and Grasp Planning in Surface Decluttering". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[125] Peter Lane Taylor. *Could 'Pandemic Drones' Help Slow Coronavirus? Probably Not-But COVID-19 Is A Boom For Business*. 2020. URL: https://www.forbes.com/sites/petertaylor/2020/04/25/could-pandemic-drones-help-slow-coronavirus-probably-not-but-covid-19-is-a-boom-for-business/.

[126] SoftBank Robotics Team. *Deploy Autonomous Cleaning Robots to Fight COVID-19 in Healthcare Facilities*. URL: https://usblog.softbankrobotics.com/deploy-autonomous-cleaning-robots-to-fight-covid-19-in-healthcare-facilities.

[127] Moritz Tenorth et al. "The roboearth language: Representing and exchanging knowledge about actions, objects, and environments". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 1284–1289.

[128] Nan Tian et al. "A Cloud-Based Robust Semaphore Mirroring System for Social Robots". In: *learning* 12 (), p. 14.

[129] Nan Tian et al. "A cloud robot system using the dexterity network and berkeley robotics and automation as a service (Brass)". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 1615–1622.

[130] Nan Tian et al. "A Fog Robotic System for Dynamic Visual Servoing". In: *arXiv preprint arXiv:1809.06716* (2018).

[131] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 23–30.

[132] Roger Tsai. "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses". In: *IEEE Journal on Robotics and Automation* 3.4 (1987), pp. 323–344.

[133] Axel Vick et al. "Robot control as a service—Towards cloud-based motion planning and control for industrial robots". In: *Robot Motion and Control (RoMoCo), 2015 10th International Workshop on*. IEEE. 2015, pp. 33–39.

[134] Markus Waibel et al. "A world wide web for robots". In: *IEEE Robotics & Automation Magazine* 18.2 (2011), pp. 69–82.

[135] Stefan Waldherr, Roseli Romero, and Sebastian Thrun. "A gesture based interface for human-robot interaction". In: *Autonomous Robots* 9.2 (2000), pp. 151–173.

[136] Chen Wang et al. "Densefusion: 6d object pose estimation by iterative dense fusion". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3343–3352.

[137] HP Wang, Y Tian, and N Christov. "Event-triggered observer based control of networked visual servoing control systems". In: *Journal of Control Engineering and Applied Informatics* 16.1 (2014), pp. 22–30.

[138] LEEE Weiss, ARTHURC Sanderson, and CHARLESP Neuman. "Dynamic sensor-based control of robots with visual feedback". In: *IEEE Journal on Robotics and Automation* 3.5 (1987), pp. 404–417.

[139] J. Weisz and P.K. Allen. "Pose error robust grasping from contact wrench space metrics". In: *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)'12*. 2012.

[140] Jonathan Weisz and Peter K Allen. "Pose error robust grasping from contact wrench space metrics". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 557–562.

[141] *What is RoboEarth?* URL: http://www.roboearth.org/what-is-roboearth.

[142] Jeff Wiegley, Anil Rao, and Ken Goldberg. "Computing a Statistical Distribution of Stable Poses for a Polyhedron". In: *In 30th Annual Allerton Conf. on Communications, Control and Computing*. 1992.

[143] Haiyan Wu et al. "Cloud-based networked visual servo control". In: *IEEE Transactions on Industrial Electronics* 60.2 (2013), pp. 554–566.

[144] Wenzhen Yuan, Siyuan Dong, and Edward H Adelson. "Gelsight: High-resolution robot tactile sensors for estimating geometry and force". In: *Sensors* 17.12 (2017), p. 2762.

[145] Chen Zhang et al. "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.11 (2018), pp. 2072–2085.

[146] Wei Zhang, Michael S Branicky, and Stephen M Phillips. "Stability of networked control systems". In: *IEEE Control Systems* 21.1 (2001), pp. 84–99.

[147] Cezary Zieliński et al. "Reconfigurable control architecture for exploratory robots". In: *Robot Motion and Control (RoMoCo), 2015 10th International Workshop on*. IEEE. 2015, pp. 130–135.