

Copyright  
by  
Rusty Alexander von Sternberg  
2016

**The Thesis Committee for Rusty Alexander von Sternberg  
Certifies that this is the approved version of the following thesis:**

**GCCF: A Generalized Contact Control Framework**

**APPROVED BY  
SUPERVISING COMMITTEE:**

<b>Supervisor:</b>	_____
	Sheldon Landsberger
<b>Co-Supervisor:</b>	_____
	Mitch Pryor

# **GCCF: A Generalized Contact Control Framework**

**by**

**Rusty Alexander von Sternberg, B.S.M.E**

**Thesis**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Master of Science in Engineering**

**The University of Texas at Austin**

**May 2016**

## **Acknowledgements**

I would like to thank my advisors, Mitch Pryor and Sheldon Landsberger, and the rest of the NRG research group for their help and support with this work. I would also like to thank my family, friends, and especially my fiancé, Sarah Heptig, for their support.

## **Abstract**

### **GCCF: A Generalized Contact Control Framework**

Rusty Alexander von Sternberg, M.S.E

The University of Texas at Austin, 2016

Supervisors: Sheldon Landsberger, Mitch Pryor

The field of robotics has come a long way since the first reprogrammable robot was able to automate simple tasks on an assembly line. However, many industrial robots are stuck doing similar simple tasks in the field, especially in the nuclear industry. Roboticists can expand the task space of industrial robots by making advanced robot technology reliable, easily integrated, and packaged in a manner that does not require an expert in the field to use. One particular field of robotics that could be used to help this task space expansion is compliant control which is used to execute robotic procedures involving contact with environmental objects. It is especially useful when the position or orientation of the environmental objects is not precise. Examples of industrial procedures that a robot could do with compliant control include material reduction, surface finishing, packaging, assembly, material handling, and many more.

This thesis explores the state of the art in compliant control and proposes a Generalized Contact Control Framework (GCCF) that packages compliant control laws in a manner that is easy to use for a non-expert. GCCF splits the control of a robot end effector into separate control of each linear and rotational dimension. The user sets the law that

controls each dimension independently to one of three intuitive laws. By specifying laws and stiffness independently for each dimension of end effector control, the user can complete a large variety of contact tasks.

We illustrate GCCF's broad capabilities in two flexible demonstrations. The first demonstration provides a graphical user interface to GCCF with which a user can set and reconfigure the control of the end effector while interacting with the robot. This allows the user to subjectively experience the reconfigurability as well as the physical behavior prompted by the control. In the second demonstration, we use GCCF to execute multiple contact tasks with the goal of putting a peg in a hole. These demonstrations prove the feasibility and usefulness of GCCF, using the API and ROS compatible package for the controller.

## Table of Contents

List of Figures .....	x
Chapter 1 Introduction .....	1
1.1 Hurdles to Industrial Automation .....	4
1.1.1 Hardware and/or Software Configurability .....	5
1.1.2 Automated Path Planning and Collision Avoidance.....	5
1.1.3 Reliable, Reconfigurable End Effectors .....	6
1.1.4 Integrated Vision Sensing .....	7
1.1.5 Integrated Force Sensing and Control.....	7
1.1.6 Current Industrial Robot Infrastructure .....	7
1.2 Task Statement and Objectives .....	9
1.3 Motivating Examples .....	10
1.4 Organization of This Thesis .....	13
Chapter 2 Literature Review .....	15
2.1 Overview of Manipulator Control Schemes .....	15
2.2 Compliant Force Control .....	20
2.2.1 Formalizing the Problem of Compliant Control .....	21
2.2.2 Hybrid Position/Force Control.....	22
2.2.3 Impedance Control.....	23
2.2.4 Bridging the Gap Between Hybrid Control and Impedance Control .....	25
2.3 Recent Force and Compliant Control Efforts .....	25
2.4 Notable Applications in Nuclear and Hazardous Environments .....	27
2.5 Summary of Literature Review.....	32
Chapter 3 Generalized Contact Control Framework Design .....	34
3.1 Design of the Generalized Contact Control Framework.....	34
3.1.1 Control Dimensions and Reference Frames .....	35
3.1.2 Control Laws.....	38
3.1.3 End Conditions.....	43

3.1.4 Examples of Setting up the Control Framework on Multiple Axes .....	43
3.2 Design Summary.....	47
Chapter 4 Implementation.....	48
4.1 Required Hardware .....	48
4.2 ROS.....	49
4.2.1 ROS Plumbing .....	49
4.2.1.1 ROS Nodes and Messages .....	49
4.2.1.2 ROS Core .....	51
4.2.1.3 ROS Packages .....	51
4.2.2 TF: ROS Package for Managing Cartesian Coordinate Frames .....	52
4.2.3 MoveIt!: Motion Planning and Robot Description .....	53
4.3 C++ Implementation .....	54
4.3.2 Safety in the Implementation .....	59
4.3.3 Other in House Code Supporting the Framework .....	62
4.3.3.1 NetFT Utilities Package .....	62
4.3.3.2 Move Interface Package.....	63
4.4 Putting it all Together .....	63
4.5 Final Notes on Implementation.....	65
Chapter 5 Demonstrations of the Generalized Contact Control Framework .....	67
5.1 Hardware Integration for Demonstration .....	67
5.2 Dynamically Reconfigurable Demonstration .....	71
5.3 Application Demonstration .....	73
5.3.2 Demonstration Tasks .....	75
5.3.3 Execution of the Demonstration Tasks .....	76
5.3.3.2 Execution of Task 1 .....	78
5.3.3.3 Execution of Task 2 .....	80
5.3.3.4 Execution of Task 3 .....	81
5.4 Summary of Demonstrations .....	83



Chapter 6 Conclusion and Future Work .....	85
6.1 Research Summary .....	85
6.2 Recommendations for Future Work.....	86
6.2.1 Position Commanding.....	86
6.2.2 More Control Laws .....	87
6.2.2.1 Explicit Force Control Law for Tasks that Require a Precise Applied Force.....	87
6.2.2.2 Adaptive Control Law.....	88
6.2.2.3 Other Variations of Impedance Control Laws .....	88
6.2.3 Modular Control Law Editor .....	88
6.2.4 Adopting the Control Framework to Other Challenging Applications .....	89
6.2.5 Task Space Formalisms .....	91
6.2.6 Additional Hurdles.....	91
6.3 Concluding Remarks.....	91
Appendix A.....	92
Appendix B .....	93
References.....	94

## List of Figures

Figure 1.1 Boston Dynamics Atlas robot [3].....	2
Figure 1.2 RVIZ planning plugin used to control through MoveIt! .....	4
Figure 1.3 Robotiq 3 finger gripper .....	8
Figure 1.4 A worker performing a procedure using the glove ports of a glovebox [11] .....	11
Figure 1.5 Autonomous material reduction solution [12].....	12
Figure 2.1 Operator teaching the Unimate [15].....	16
Figure 2.2 Memory drum recording Unimate commands [15].....	17
Figure 2.3 Block diagram of force feedback control method [16] .....	18
Figure 2.4 Types of Manipulator Control .....	20
Figure 2.5 Robotic manipulator constrained by pump C-surface. $v$ is the linear velocity. $\omega$ is the rotational velocity. $F$ is the linear force. $\tau$ is the torque. $C$ is the constrained velocity. $f(v)$ is a function of velocity and the dynamics of the system. ....	22
Figure 2.6 SADIE robot [45] .....	29
Figure 2.7 Fanuc arms in an ARIES glovebox [43].....	31
Figure 3.1 Design interfaces .....	35
Figure 3.2 Managed reference frames. The FT frame is the frame that the force/torque data is read in. The robot control frame is the frame that the framework sends robot commands in. The task frame is the frame that the user specifies control task in.....	36
Figure 3.3 Peg in hole problem using follower law.....	39
Figure 3.4 Surface follower problem using spring law.....	40

Figure 3.5 Guarded move problem using compliant move law .....	41
Figure 3.6 Using the control framework with uncertain orientation .....	45
Figure 3.7 Screwing on a lid with the generalized contact control framework .....	46
Figure 4.1 ROS plumbing .....	52
Figure 4.2 Class diagram of generalized contact control framework .....	55
Figure 4.3 Typical flow of contact task procedure .....	57
Figure 4.4 NRG Safety Architecture .....	61
Figure 4.5 Implementation and Interfaces of the generalized contact control framework .....	64
Figure 5.1 ATI Gamma FT transducer [54] .....	68
Figure 5.2 Motoman SIA5 7DOF Robot .....	69
Figure 5.3 Demonstration Hardware Setup and Integration .....	70
Figure 5.4 Robotiq 3-Finger Gripper [58] .....	71
Figure 5.5 Dynamically Reconfigurable GUI. Each Axis has a drop down menu that allows the user to select desired control law (None, Move, Spring, or Follow) well as modify the stiffness parameter using a simple slider.	72
Figure 5.6 Peg in hole demonstration configuration .....	74
Figure 5.7 Tasks for peg in hole application demonstration: 1) grasping the peg, 2) balancing and re-grasping the peg, 3) tracking to the hole and inserting the peg .....	75
Figure 5.8 MoveIt! planning scene including collision objects .....	77
Figure 5.9 Peg in hole demonstration tasks .....	78
Figure 5.10 User defined coordinate reference frame for task 1 moves .....	79
Figure 5.11 User defined coordinate reference frame for task 2 moves .....	81
Figure 5.12 User defined coordinate reference frame for task 3 moves .....	82

Figure 6.1 A position controller implementation.....	87
Figure 6.2 Future applications of GCCF. Top: A robot size-reduces a bowl [12]. Middle: Two robots hold an egg [59]. Bottom: A robot opens a cabinet door [60].....	90

# **Chapter 1**

## **Introduction**

Robots have been able to do simple tasks such as material handling, welding, and painting as early as the 1960s and 1970s [1]. In fact, the Unimate, the first reprogrammable industrial robot used on a large scale assembly line, was patented in 1954 [2], and was first used by General Motors in 1961 to move and weld parts on an assembly line [1]. The Unimate revolutionized simple automation in assembly line tasks, allowing a factory operator to use a single robot on multiple, short run tasks rather than having to design a fixed automation solution for each task. Since these early days of robotic automation, robots have developed a wide array of new skills such as teleoperation, automated path planning, collision avoidance, learning, and integration with vision sensors and force and torque sensors.

With these new skills, along with staggering advances in computational power, robots have accomplished amazing new feats. Boston Dynamic's Atlas robot, seen in Figure 1.1, walks on two legs along rugged terrain, and responds to forceful disturbances such as being pushed over by a human [3]. Quadcopter drones fly above us while surveying land and relaying video streams and other sensory information back to the ground in real-time. Soon, they might even deliver packages [4] or capture other flying drones [5].



Figure 1.1 Boston Dynamics Atlas robot [3]

With the amazing advances made in robotics in the last few decades, industrial automation, and especially in the nuclear industry, has fallen behind given the rapid pace of advancement. Ideally, a robot on an assembly line or in an industrial setting would do all the tasks that a human could complete, but with better precision, for longer hours, and without causing injury to humans or the objects it manipulates. An ideal industrial robot must also have the flexibility of a human to move from one task to another. As Unimate proved, by allowing the robot to be reprogrammable and reconfigurable for a specific task, robotic automation becomes viable for not only tasks that will be completed for long periods of time, but also short run tasks.

With the advances seen in robotics, why are so many industrial automation systems still doing the simple tasks they have done for years? Why is industry not closer to having the ideal, flexible robot that can replace or augment humans on the assembly line? This failure of industry to accept new technological advances is seen in many disciplines and is often referred to as “traversing the valley of death” [6] [7]. In the case of industrial automation, this “valley of death” is often caused by a lack of packaging of robotic technologies. It takes a roboticist or even an expert in a specific field of robotics to assemble all the pieces that are required to complete each specific task. If the pieces were reliable and packaged in a way that a non-expert could assemble and use them, then industry could adopt new robotic technologies.

An example of successful packaging of pieces of advanced technology in a way that can be used by a non-expert is MoveIt! [8]. MoveIt! will be described in greater detail in Chapter 4, but on a high level it is a software package that a user utilizes to move a robot with automated path planning and collision avoidance. MoveIt! incorporates many advanced pieces including:

- **Kinematic calculations** – The equations of motion of the links of the robot that are used to map joint motions and torques to tool point motions and forces.
- **Path generation** – The generation of joint trajectories (a queue of robot joint positions).
- **Path selection** – The selection of the best, collision-free path from a list of randomly generated trajectories.
- **Collision detection and avoidance** – The ability to model the environment in a *collision scene* and detect which generated paths result in collisions.

With the simplest version of the interface to MoveIt!, pictured in Figure 1.2, a user drags a virtual robot to a new position and presses a “plan and execute” button to safely move the robot while avoiding collisions with the modeled environment. The grey robot pictured in Figure 1.2 is the current position of the actual robot, and the orange robot is the desired position for the next move. The user drags the orange robot to a desired position using the arrows attached to the last link of the robot. The environment can be modified by uploading standard format CAD (Computer-aided Design) models, or specifying dimensions of simple shapes using the same user-friendly interface.

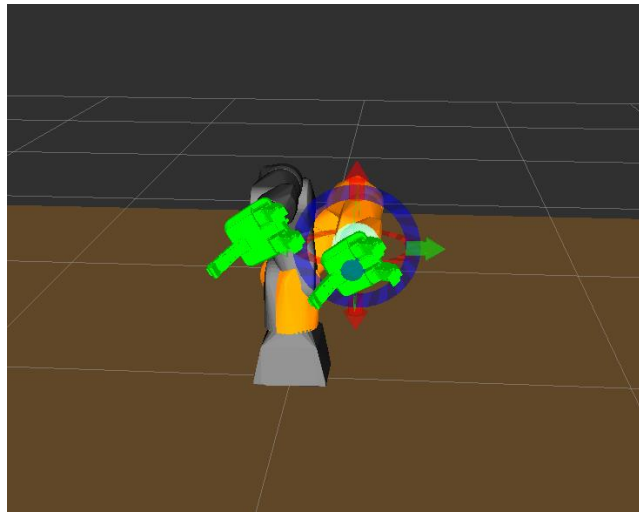


Figure 1.2 RVIZ planning plugin used to control through MoveIt!

## 1.1 HURDLES TO INDUSTRIAL AUTOMATION

The simplicity and configurability shown by the packaging of MoveIt! is what advanced robotics capabilities need to be accepted by industry, but there are more aspects of advanced robotic control that still need to be made reliable and packaged in this way. Not only do the pieces need to be packaged, but an overall infrastructure must be created that can include and manage the packages so they can be used together. This infrastructure



will need to include a variety of hardware and software capabilities necessary to close the “flexibility gap” between operators and automation. Each of these (and the related technical challenges) are discussed below, but include:

- Hardware and/or software configurability
- Automated path planning and collision avoidance
- Reliable, reconfigurable end effectors (EEFs)
- Integrated vision sensing
- Integrated force sensing and control

This section will briefly review a few of the core components of an advanced robotic infrastructure, focusing on concepts that are important to the nuclear industry and this thesis.

### **1.1.1 Hardware and/or Software Configurability**

In order for the infrastructure to be reconfigurable, it must have standard hardware interfaces so that tools and parts can be changed and replaced easily. This concept is well understood and has been seen in mass production as far back as the original Ford assembly line. Possibly even better than hardware configurability, is improved software that maximizes the number and complexity of tasks that hardware can perform without reconfiguration. In the nuclear industry, this is even more important. Robots operate in contaminated and/or sealed environments. Reconfiguring hardware requires decontamination or other activities that lead to costly downtime.

### **1.1.2 Automated Path Planning and Collision Avoidance**

Manipulator paths can be preplanned for simple pick and place operations with static environments, but when robots complete multiple tasks with undefined parameters, they must generate preferred motion plans autonomously. When generating this path, or

motion plan, the robot should take into consideration avoidance of known environmental obstacles and its own limits, e.g. travel limits, velocity limits, singularities. The position of the environmental obstacles should be configurable via a software model. This could be the job of the human operator, or this information could come from sensing capabilities. Path planning processes must happen in a reasonable amount of time to assure the task completes in reasonable amounts of time. A key advantage of using automation is the reduction of exposure and the risk of injury by the operator, but these advantages cannot come with significantly increased production times or development costs.

### **1.1.3 Reliable, Reconfigurable End Effectors**

EEFs are the tools attached to the last link of a serial manipulator. In order to accomplish the goals of a robotic infrastructure that is flexible, EEFs must accomplish many different tasks. For an industry that does not put robots in a dangerous environment, this might mean EEFs that have an adapter between the last link of the robot and the EEF that makes it easy to switch between different EEFs specifically designed for a task. Or it might mean a design that allows the robot itself to change the EEF. For the nuclear industry, when the robot might be in a dangerous environment, one EEF that is able to accomplish multiple tasks would be the ideal situation. This would allow the least amount of hardware reconfiguration and therefore less chance that an operator would need to physically adjust the robot.

Another important goal for EEFs is that they need to be reliable. Grasp validation, or checking that an EEF has successfully picked up an object, will be important for a robot to be able to run independently and autonomously.

#### **1.1.4 Integrated Vision Sensing**

Integration with vision sensors will be necessary to determine changing environments to update the software model. Also, vision sensors can locate objects to be manipulated, sort or classify objects, and validate that tasks have been completed successfully. All of this will be necessary to create a reliable, non-task-specific platform.

#### **1.1.5 Integrated Force Sensing and Control**

While vision sensing can provide pose estimation and location of environmental objects, the position provided will always be an estimate that relies on the precision and calibration of the sensor. To handle the uncertainties in vision estimates, and others, integration with force and torque sensors will also be important to the platform. To do any task in which contact with the environment is necessary, force sensing is required. When contacting with the environment, small deviances in the model will lead to dangerously high contact forces. Therefore, the robot must have the ability to comply with the physical restrictions in position that the environment imposes. In order to execute multiple different tasks, the robot's compliance should be adjustable or general enough that it can deal with different environments.

#### **1.1.6 Current Industrial Robot Infrastructure**

Some of the previously mentioned components of a flexible, safe robotic infrastructure exist. For example, many industrial robot controllers perform kinematic calculations and control each actuator to move a robot from one position to another. These controllers typically send the joint positions or trajectories to the joint actuators. However, industrial controllers do not always include automated path planning and collision avoidance described previously. But reliable algorithms and code libraries, including

MoveIt!, do exist to handle these tasks, some of which will be discussed in the Chapter 4 of this thesis.

In the category of reliable, reconfigurable EEFs, tool changers allow robots to change their own tools autonomously [9]. Recently developed sensors allow for a robot hand to feel objects to validate grasps [10]. The ability to use a variety of EEFs allows for many types of tasks to be completed, but an EEF as universal as the human hand has not been achieved yet. Two and three finger grippers, such as the one pictured in Figure 1.3, offer a combination of flexibility and robust operation, but still have their limits. For example, without force sensing, the changing length of the fingers as they close can make it difficult to pick up small objects. Another common issue is the bulkiness of grippers combined with their necessary control components. Also the necessity to power and communicate with the gripper can cause difficulties since extra wiring can hinder robot motion.

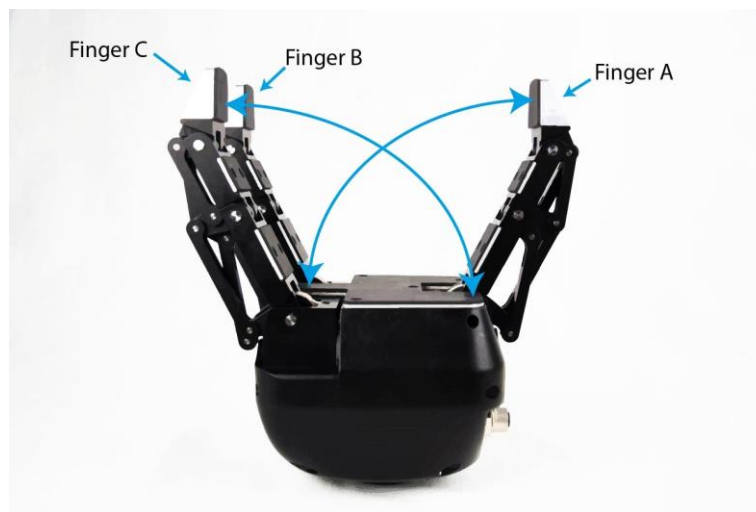


Figure 1.3 Robotiq 3 finger gripper

Vision sensors can learn the signature of objects and find them in a scene. They can determine the pose of the object to assist a robot to grasp it or avoid colliding with it. They can also analyze a scene to see what has changed or read barcodes on objects.

Control algorithms have been developed to use force sensor data to manipulate environmental objects. They can also validate grasps and stop operation under unsafe conditions. For select individual tasks, force sensing and control has been addressed and some key examples are discussed in Chapter 2, but little has been done to reduce the burden on the developer when new contact tasks are considered. So, while possible, this capability is largely missing from the manipulator's general supporting infrastructure.

Although many of the solutions to the discussed hurdles are mature in some sense, not many of them have been accepted into the industrial community. This might be due to a lack of integration of the components, and the difficulty of taking all the components and creating a robotic procedure. At the current state of maturity, many of these solutions to the hurdles discussed require an expert in the field to apply the solution. In order to expand the task space of robots in the field, roboticist must mature this robotic infrastructure and make it available in a way that does not require expertise to use.

## **1.2 TASK STATEMENT AND OBJECTIVES**

The section above outlines some of the hurdles that roboticist must be address and integrate to broaden the task space of industrial applications of robotics. The Nuclear Robotics Group at UT Austin is concurrently addressing many of these issues, but this effort attempts to tackle issues related to integrated force sensing and control. A generalized framework for compliant control, GCCF, is proposed. The primary objectives of the framework design are:

- **Flexibility** – Able to be used on a large number of contact tasks

- **Safety** – Use force sensing to add a level of safety to infrastructure
- **Simplicity** – Able to be used by non-expert
- **Hardware Agnostic** – Able to run on many hardware platforms
- **Ease of integration** – Easy to hook up to existing infrastructure
- **Modular/Extensible** – Allow for improvements/extensions

GCCF generalizes the contact control process and provides a few simple control laws to perform contact control. This generalization ensures flexibility for a variety of tasks. GCCF also uses force sensing to add a layer of safety to a robotic procedure that can compensate for non-precise environments. GCCF and its associated control laws are kept simple and use intuitive concepts so that they do not require an expert in the field to understand them. GCCF is also hardware agnostic and thus compatible with multiple platforms. Along the same lines, it is easily integrated using existing robot infrastructure code. GCCF is modular and extensible so that users and/or researchers can easily improve upon the design as they experience future challenges.

### 1.3 MOTIVATING EXAMPLES

GCCF could be useful for a wide variety of applications in industry. One such application at Los Alamos National Lab (LANL), is material reduction. Engineers at LANL need to size reduce plutonium pits to fit them into a standard crucible in which they are melted down to be recycled. Currently, workers perform procedures like this procedure by manipulating objects through gloves of a glove-box, as seen in Figure 1.4.

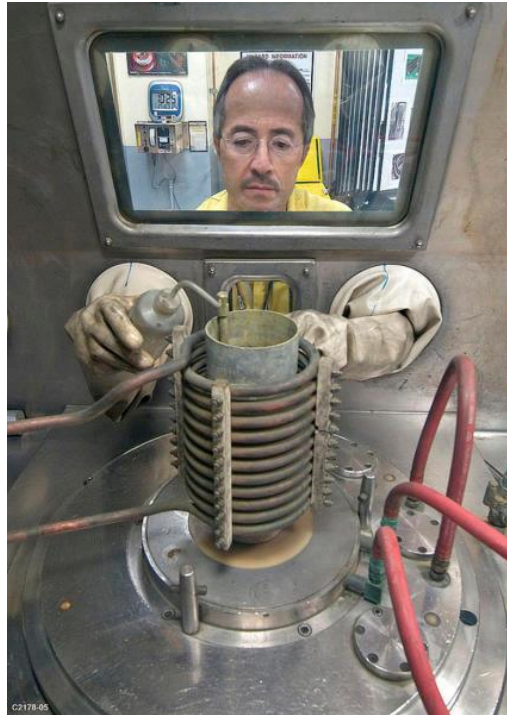


Figure 1.4 A worker performing a procedure using the glove ports of a glovebox [11]

Since the size reduction procedure generates small metal scraps, it is possible for the metal to cut the gloves exposing the worker to the dangerous environment within the glovebox. A recent NRG student demonstrated an autonomous material reduction procedure that successfully completed this task (with a metal bowl, instead of a plutonium pit) using a robot with a vacuum gripper EEF shown in Figure 1.5. The robot picks up the bowl and places it in the hole punch which is automatically actuated. The robot then continues to rotate the bowl in the punch until it is sufficiently reduced in size.



Figure 1.5 Autonomous material reduction solution [12]

The vacuum gripper chosen for the demonstration is inherently compliant and alleviates stresses that the hole punch puts on the manipulator. However, a vacuum system might not be the most appropriate EEF since pieces of the material could get caught in the suction of the vacuum or the gripper's rubber (i.e. organic) material which is problematic. Thus, a stiffer EEF may be chosen and in that case there would be need for a compliant controller. An EEF that rigidly holds an object in the punch could react to the force applied by the punch via GCCF. This type of behavior would be difficult to hard code into the procedure since the reaction will be different at the differing radii of the bowl, but with a controller, like the one proposed, a robot is able to account for differing contact forces. Since GCCF has adjustable stiffness, fine tuning would allow the EEF to be stiff enough that the punch is executed properly, and compliant enough that the robot is not improperly stressed. The framework would also be useful when picking up the object, if the position is not known precisely. It is possible that a human operator would be placing the pits in



front of the robot to perform this operation. In this case the pit would not be in a precise position and it would be unsafe to pick them up without force monitoring and control.

Other possible applications include material handling, surface finishing, station scheduling, and co-robotics. As was stated in the previous example, material handling in imprecise conditions is a prime example of the usefulness of GCCF. Not only could GCCF help a robot pick up an object with imprecise knowledge of its position, but after the robot picks up the object, force monitoring could be used to validate the grasp and GCCF to move the object safely away from the environment. Station scheduling is using the same glovebox setups for multiple tasks. This is where software, instead of hardware, configurability becomes very useful. Since GCCF is software configurable and does not require swapping of EEFs, workers can decide which tasks are to be done in a glovebox without having to swap out the hardware in the glovebox. Workers might also work alongside robots someday. In co-robotic applications, it is important for worker safety that EEFs are compliant when humans are interacting with the robot. GCCF could also handle this type of behavior.

#### **1.4 ORGANIZATION OF THIS THESIS**

This chapter introduces the hurdles prohibiting expansion of reusable industrial automation and the objectives for tackling the hurdle of integration of force sensing and control. It proposes a generalized contact control framework, GCCF, and briefly described some motivating examples in which GCCF would be useful. The remaining chapters are organized as follows:

**Chapter 2** shows a progression of robotic control into the compliant control field and summarizes previous work and the state of the art in the field. It also gives an overview of current state-of-the-art robotic control in the nuclear industry.

**Chapter 3** details the design of a generalized contact control framework that adheres to the objectives laid out in this chapter.

**Chapter 4** describes the implementation of GCCF in C++ using the Robot Operating System, ROS.

**Chapter 5** shows two demonstrations of GCCF that are relevant to the motivating applications outlined above. The first demonstration is a graphical user interface that allows the user to experiment with the workings of the framework. The second demonstration shows GCCF being used to accomplish a robotic procedure involving multiple contact tasks.

**Chapter 6** concludes this thesis and suggests avenues for future work to extend GCCF capabilities.

## **Chapter 2**

### **Literature Review**

GCCF's underlying algorithms are based largely on previous work in the field of compliant robotic control. The proposed framework synthesizes multiple ideas that have been proven in the past and implements them in a general and hardware agnostic manner. This literature review takes a look at robotic control in the literature, giving attention to control involving interaction with the environment and nuclear industry applications.

First, a general overview of manipulator control architectures is presented before narrowing the focus to methods utilized to control contact with the environment. Historical progression from explicit force control to more modern methods of controlling forces and positions simultaneously are presented as well as a look at recent advances in the literature. Although non-contact control is used in conjunction with the framework, the origins of this control will not be discussed in this literature review. This is because the framework uses proven and easily accessible methods to control the manipulator when not in contact with the environment. These methods incorporate kinematic calculations, motion trajectory planning, and collision avoidance and are achieved via the Robot Operating System, ROS, [13] and will be discussed in the implementation chapter. Lastly, applications of robotics in the nuclear industry are considered and the current state of the art and innovative applications in the industry are reviewed.

#### **2.1 OVERVIEW OF MANIPULATOR CONTROL SCHEMES**

The first form of reprogrammable robotic control in an industrial robot was the Unimate robot, patented in 1954 [2], and was used by General Motors for spot welding [14] and other tasks. Although machines capable of completing assembly line tasks existed before the Unimate, they were typically designed for a single task and therefore only

feasible for tasks that ran long term. The Unimate, on the other hand, could be “taught” to learn a new task without reconfiguration of the robot. This teaching process did not require a skilled technician to reprogram the Unimate. To teach the Unimate, an operator would move the robot to each position required to complete a task by using a simple control box seen in Figure 2.1.

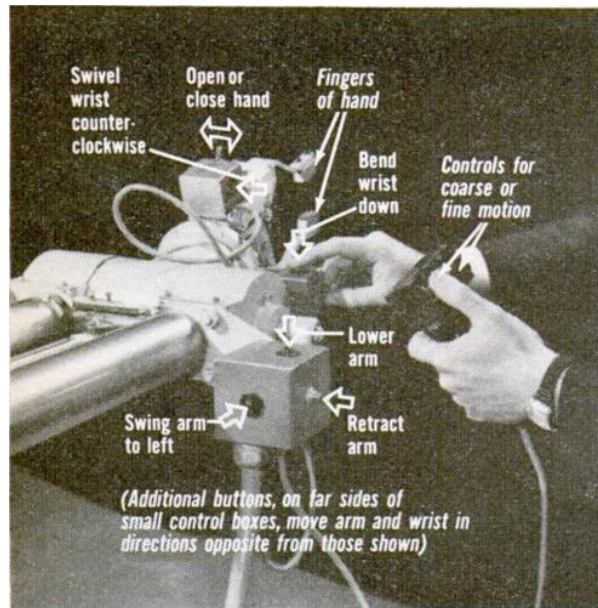


Figure 2.1 Operator teaching the Unimate [15]

The operator would then press the record button to record the position on a 30-inch-long magnetic drum, shown in Figure 2.2, which could store up to 200 positions or commands. Besides moving to a position, the commands could also delay the next move or open or close the gripper. When a commanded position is played back, the robot would move its links with separately controlled hydraulic actuators until all position encoders read the right positions [15].

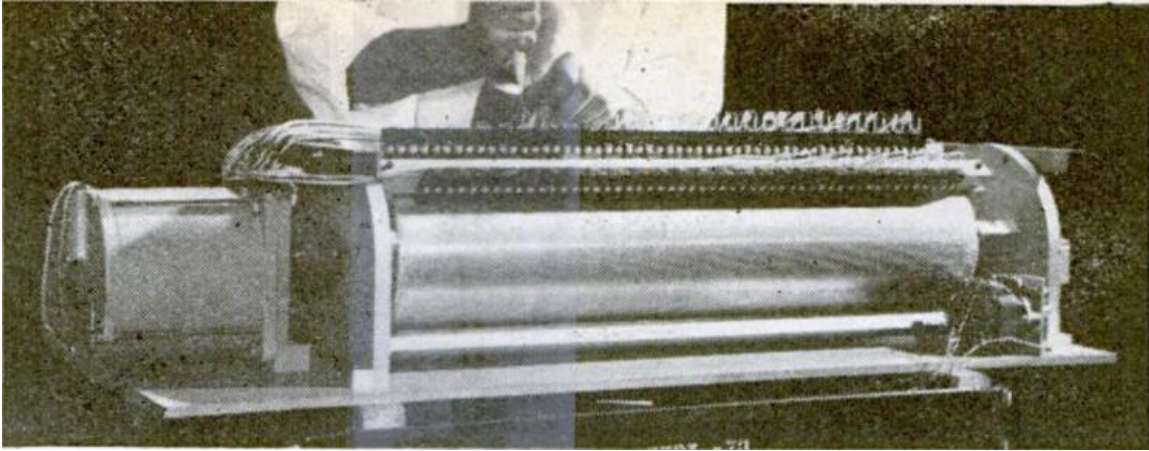


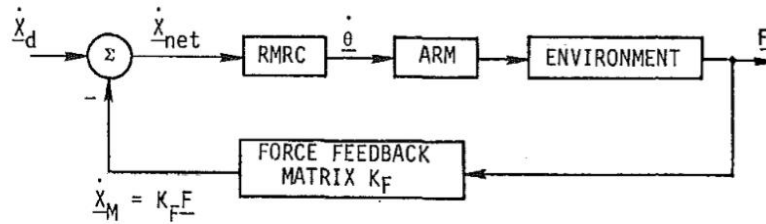
Figure 2.2 Memory drum recording Unimate commands [15]

This process of “teaching” a robot by recording positions is still used in the industry today, but more advanced position control schemes have also been developed. In modern robotic control schemes, wrapped around the joint position controllers is typically a path planner that determines a stream of appropriate joint position commands, or a trajectory, that will take the robot from one set of joint positions to another. While generating this trajectory, the path planners also avoid known obstacles in the environment and positions in which control laws break down or behave undesirably, e.g. at travel limits or near singularities.

The Unimate and many other industrial robots solely used position commanding to complete its task and do not take into consideration other important information that can be retrieved via sensors, e.g. cameras or force sensors. For more complex tasks, this sensory information can be used to avoid barriers to motion that have been introduced to the robot’s workspace, or to compensate for uncertainty. This lack of sensory information makes pure position commanding useful only for operations where the manipulator has minimal contact with the environment. Once there is contact, slight deviations in a robot’s desired position and its actual position, or deviations between the model and actual environment,

lead to dangerously high contact forces. For this reason, a robotic system must take into account model inaccuracies by sensing contact forces on the robot. This problem spawned the field of force control which commands the robot based on sensed forces. According to Whitney [16], one of the pioneers of force control, “gross motions”, i.e. open loop position control motions, are useful for “material handling tasks as well as ‘assembly’ tasks such as spot welding in which insertions of one part into another are not necessary”, but fine motions, i.e. closed loop motions based on force feedback, “are required for some types of assembly requiring insertions, push and twist actions, gear meshing, packing, and so on.”.

The simplest way to apply force control is to command robot joint positions only based upon the knowledge of current and desired EEF contact forces. This can be useful in applications where a precise force is desired to be maintained. Such a method was proposed by Whitney and termed linear force feedback strategy. In this method, a force feedback matrix, a matrix of feedback gains, is used to convert sensed forces into desired EEF position deltas which are then converted into new joint position commands [16]. This process is illustrated in Figure 2.5.



**Fig. 2 Accommodation servo**

Figure 2.3 Block diagram of force feedback control method [16]

This literature review will focus on control algorithms that contain strategic elements of both position and force control methods. This middle ground is the area of

active compliant control. Active compliant control attempts to track a trajectory, i.e. position control, while maintaining compliance with respect to physical contact (intentional or not).

It is worth mentioning that there is another area of compliant control called passive compliant control. Active compliant control is attained through software whereas, passive compliant control requires that the robot or EEF is inherently mechanically compliant. Passive compliance is achieved using a spring, clutch or other compliant device between the EEF and last link of the robot manipulator [17], in the EEF itself [18], or in each actuator [19] [20]. In passive compliant control, the robot is position controlled, and the compliance of the robot itself allows for the contact forces to be minimized. While variable stiffness actuators have been investigated, [21], no physical system was identified that would give a passive controlled robot the range of performance in terms of precision, payload, etc. necessary for the applications proposed by the sponsor. Furthermore, active compliant control techniques can be implemented on proven affordable industrial manipulators, whereas multi-purpose passive compliant control architectures, exemplified by Rethink Robotics' Baxter [22], have only been available on the market for a few years, and none have the payload or precision requirements necessary for the envisioned applications in the nuclear domain.

Figure 2.4 shows the types of manipulator control. In the domain between explicit position control and explicit force control, there are two main philosophies of control that will be discussed. One of them I have termed split control, where the control method attempts to use explicit force and explicit position control separately in different Cartesian directions so that the robot may reach a desired position while also being compliant. The other, I have termed relational control. In this type of control, the control method enforces a dynamic relation between the position of the EEF and contact forces in all directions. In

between these two types of control there is hybrid impedance control which attempts to synthesize the benefits of both types of compliant control.

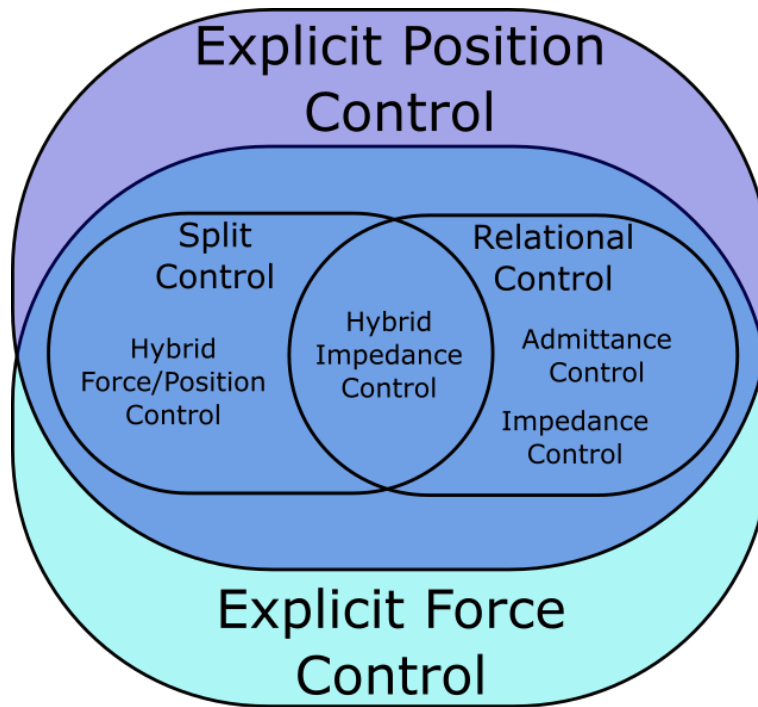


Figure 2.4 Types of Manipulator Control

## 2.2 COMPLIANT FORCE CONTROL

To manipulate objects in the environment or to mitigate accidental contact with the environment, robotic control must implement more than just simple position control, it must also control the contact forces between the EEF and the environment. Force control uses sensory force and torque information to adjust the position of an EEF to maintain or mitigate forces at the contact with the environment. Explicit force control relies solely on force data to calculate the desired robot movement that will maintain a desired force. This type of control can be very useful in certain applications, but its fault is that it pays no attention to a desired position of the EEF. In a lot of cases, the operator would like to



maintain a desired force profile while also staying as close as possible to the desired position of the EEF.

### **2.2.1 Formalizing the Problem of Compliant Control**

Mason formalized the problem of controlling contact forces while maintaining a desired position in 1981 as *compliant motion*. According to Mason, “compliant motion occurs when the position of the manipulator is constrained by the task” [23]. Mason focused on active compliance solutions rather than passive compliance.

Mason introduced the concept of a *C-surface* which is “a task configuration space which allows only partial positional freedom” [23]. A C-surface is the intermediate between two extremes, total positional freedom and no positional freedom. While not in contact with the environment, a manipulator has complete positional freedom. On the other extreme, a manipulator rigidly attached to a stiff object has no positional freedom and has complete freedom to control the forces on the EEF. While in contact with a C-surface, a manipulator must consider both position and force control.

Mason went on to develop a method for breaking down the natural constraints of ideal C-surfaces and adding artificial constraints that the operator would like to enforce. The simplest example of this is a manipulator following the surface of a table. A natural constraint is that the velocity of the manipulator in the direction that is normal to the surface of the table must be zero since it cannot move through the table. An artificial constraint constrains the velocity normal to this natural constraint, i.e. the velocity that moves the EEF along the surface of the table, to the desired velocity. Mason’s efforts to create this language describing contact tasks was meant to allow future work of synthesizing control strategies that enforce these natural and artificial constraints.

Figure 2.5 gives a visual of a C-surface. A robotic arm is holding the handle to a pump. This is a C-surface because the robot has neither complete position nor force freedom on all axes. In this case, the pump geometry naturally constrains velocity to zero in the  $y$  and  $z$  dimensions. The pump geometry also contains rotational velocity to zero around the  $y$  and  $z$  axes. Torque around the  $x$  axis is also zero if it is assumed that the piston is free to rotate without friction within the pump. The artificial constraints shown are the desired constraints that are not necessary due to the geometry. In this case, the robot will move the pump piston with a constant velocity, seen by the artificial constraint on the  $x$  velocity. The force in the  $x$  direction will then be some function of that velocity and the dynamics of the system. The other artificial constraints show a desire to avoid contact forces where an applied force is not needed and to have no motion around rotational  $x$  axis.

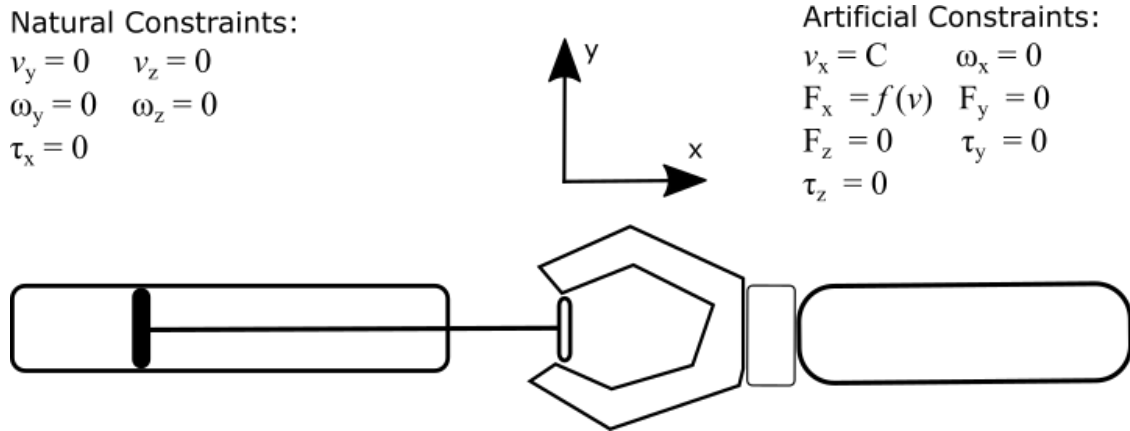


Figure 2.5 Robotic manipulator constrained by pump C-surface.  $v$  is the linear velocity.  $\omega$  is the rotational velocity.  $F$  is the linear force.  $\tau$  is the torque.  $C$  is the constrained velocity.  $f(v)$  is a function of velocity and the dynamics of the system.

### 2.2.2 Hybrid Position/Force Control

Researchers devised two methods for achieving compliant motion in the early 1980s which are still used today. First, Raibert and Craig developed hybrid position/force control [24]. Raibert and Craig took the direct logical step from Mason's formalized

constraints to synthesize a control strategy. They set up a Cartesian frame that described the natural and artificial constraints and used it to pick directions controlled by explicit position control and directions controlled by explicit force control. Then, they developed a method of transforming these control requirements to direct control of each individual robot joint. This method can be seen in equation 2.1.

$$\tau_i = \sum_{j=1}^N \{ \Gamma_{ij} [s_j \Delta f_j] + \psi_{ij} [(1 - s_j) \Delta x_j] \} \quad (2.1)$$

where  $\tau_i$  is the torque applied to the  $i^{\text{th}}$  actuator,  $N$  is the number force controlled degrees of freedom in the Cartesian reference frame plus the number of position controlled degrees of freedom in the Cartesian reference frame,  $\Delta f_j$  is the force error,  $\Delta x_j$  is the position error,  $\Gamma_{ij}$  is a force compensation function,  $\psi_{ij}$  is a position compensation function, and  $s_j$  is a binary (0 or 1) vector that indicates which degrees of freedom are force controlled [24]. The equation takes the Cartesian force and position control efforts and maps them to the torque required to be applied to each joint to accomplish the control goal. Raibert and Craig implemented the controller on a 2 axis Scheinman manipulator to show that the controller was feasible and stable. However, it is important to note that Lipkin and Duffy refuted this method in 1988 because it is “based on the metric of elliptic geometry and is thus noninvariant” with respect to Euclidean collineations and change of Euclidean unit length [25]. Lipkin and Duffy, along with others proposed new invariant hybrid position/force control methods to attempt to solve the issues reported with Raibert and Craig’s version [25] [26] [27].

### 2.2.3 Impedance Control

The literature usually groups the other method of achieving compliant motion into a category called impedance control. Though there are many different control schemes thrown into this one category, e.g. stiffness control [28] and admittance control [29], the

term impedance control comes from Hogan's work in the field [30]. Hogan pointed out that while in general absolute control of EEF position is desired, no controller can make up for the fact that the robot in contact with a physical system must behave according to the physical laws of the combined system. For this reason, the controller should command the desired motion of the manipulator, but also help it to react to disturbances that it encounters. According to Hogan, in impedance control "the controller attempts to implement a dynamic relation between manipulator variables such as end-point position and force rather than just control these variables alone" [30].

In Hogan's implementation of impedance control, he derives the following equation (2.2) for the desired relationship between the contact force and the dynamics of the system.

$$F_{int} = K(X_0 - X) + B(V_0 - V) - M dV/dt \quad (2.2)$$

where  $F_{int}$  is the "interface" force,  $K$  is an adjustable stiffness variable or nonlinear function,  $B$  is an adjustable damping variable or nonlinear function,  $(X_0 - X)$  is the difference between the commanded position and the actual position,  $(V_0 - V)$  is the difference between the commanded velocity and the actual velocity, and  $M$  is the inertia tensor of the manipulator. Therefore, according to Hogan, the equations of motion for the manipulator coupled with the environment are seen in equation 2.3.

$$\frac{(M_e + M)dV}{dt} = K(X_0 - X) + B(V_0 - V) + F_{ext} \quad (2.3)$$

where  $F_{ext}$  is the external force, or the force that the manipulator will need to apply to maintain the desired dynamic behavior, and  $M_e$  is the inertia tensor of the environment. Note that in order to implement equation 2.3, one must have precise information about the inertia of the environment and robot.

#### **2.2.4 Bridging the Gap Between Hybrid Control and Impedance Control**

While the initial efforts by Raibert, Craig, Hogan and other early pioneers of compliant control individually gave a few possible solutions to the problem of controlling force and position together, researchers eventually realized that by combining the efforts there might be an even better solution to the problem. In 1988, Anderson and Spong applied these two methods to one control strategy and called it hybrid impedance control [31]. Not only did the control combine impedance and hybrid position/force control, but it also implemented an outer/inner loop of control so that the compliant control may be done separately from the inverse dynamics calculations. Lui and Goldenberg studied this control method further in 1991. They made it more robust by the use of the computed torque technique and a PI control law to compensate for model uncertainties [32].

### **2.3 RECENT FORCE AND COMPLIANT CONTROL EFFORTS**

Since the early 2000's, a dramatic increase in computing power and expanding robot infrastructure have influenced efforts to improve force and compliant control in the literature. With the rise of faster computers, the academic community has renewed its interest in more advanced control methodologies and design that allow for greater robustness and stability, especially in cases of uncertainty. This resurgence has also included the fields of artificial learning, and neural networks. Recent literature in the field of force and compliant control schemes mirrors these advances in the state of the art of control and computing.

One issue with the original idea of impedance control is that there will be uncertainty in the robot model and especially in the model of the environmental stiffness. This issue makes it difficult to perform robust force tracking. Jung proposed a force tracking impedance control scheme that uses an adaptive control philosophy to adjust the velocity profile during motion as a function of the force error [33]. Jung showed that this

controller works well in unknown environments and when the environment stiffness is abruptly changed. Researchers have also made similar efforts for adapting to unknown parameters using neural networks. In [34], a neural network is used to adjust an impedance controller for unknown environments. In [35] a neuro-adaptive controller is used to track position and force along a flat surface with non-parametric uncertainties in the models of the robot and environment. Another innovative advancement is the use of model-free reinforcement learning and optimal control to learn variable impedance for a robotic system. Buchli [36] developed a method to allow a robot to learn variable impedance so that the robot may be compliant when able, yet stiff when required. Lee [37] took a biological approach to impedance control. By looking at the way humans interact with objects, Lee developed a control algorithm that adapts the arm stiffness based on the force error and interestingly even allows for negative stiffness.

Researchers have also made many efforts to enhance robustness of impedance control. Jin [38] used time delay estimation and ideal velocity feedback to allow for nonlinearities in robot dynamics without actually modeling them. He showed that the controller improves robustness in cases involving nonlinear friction and allows for relatively simple tuning. Another approach using time delay estimation, [39], attempts to improve robustness without sacrificing accuracy using internal model control. Another example of an effort to enhance robustness can be found in [40], where researchers attempt to improve robustness of task space impedance control on a redundant 7 degree of freedom (DOF) manipulator. Kikuuwe [41] attempted to deal with robustness in cases where the robot actuators become saturated by using a proxy-based sliding mode controller. According to the author, this saturation can result in “undesirable behaviors such as oscillation, repeated overshoots, and instability” [41].

The literature shows that compliant control is a very mature topic dating back to the late 1980's. Research has been done on both passive and active compliance, and even passively compliant devices that actively change their stiffness. Active compliance has advanced to learn and adjust compliance automatically for specific tasks. The real barrier between the academic research and industrial application is generality. While researchers have studied these learned compliant behaviors in academia and applied them to specific situations, a factory worker, or even a non-expert programmer, cannot program a commercially available robot to behave in this manner. Until roboticist develop general, non-task-specific, applications of active compliant control to be easily adopted into the current state of industrial automation, most industrial automation processes will be limited to the traditional learned position procedures that industrial automation has used since the invention of the Unimate.

#### **2.4 NOTABLE APPLICATIONS IN NUCLEAR AND HAZARDOUS ENVIRONMENTS**

It is important to look at the state of the art in the nuclear industry to see where improvements need to be made and how well the reviewed methods have been adopted to complete relevant tasks in this domain. Given the rigorous testing and safety standards required to safely operate in the nuclear industry, it is possible for the state of the art in the industry to fall behind not only the academic/research community, but also the broader automation industry as well. Oddly enough, it is the same work dangers driving these increased requirements in safety that begs the need for robotics to be developed that can aid humans in the nuclear field. In this section, literature describing robotics and automation of tasks in nuclear facilities will be reviewed, paying attention to force/torque control capabilities of these systems.

Robots are especially useful in the nuclear industry in cases where there is radioactive material and radiation present. It is a commonly held philosophy that radiation dosage to workers should be ALARA (As Low As Reasonably Achievable) [42]. With the use of robotics in the industry, it is possible to keep the dosage for certain tasks to approaching zero. The nuclear industry has already achieved a few important tasks with robotics. Some aspects of the nuclear power and defense industries that currently involve robotics are inspection, maintenance, commissioning/decommissioning of new and old facilities, waste disposal, and glovebox operations [43] [44].

Robotic test and inspection has been used across the nuclear industry. Robots are able to inspect areas of a nuclear power plant that humans would not be able to get to during operation. This allows for power plants to remain running during inspection. Considering the plant only makes money while it is running and producing energy and that shut down and restart procedures are not as easy as flipping an on/off switch, continuous operation is important to the industry. Examples of robots for this task are pipe-crawling robots that inspect pipes for cracks, the Trans-world Reactor Vessel Examination System (TWS) that is used to inspect different kinds of reactor vessels across the world, and snake-arm robots used for inspection of leaks [44]. For the most part, inspection robots are limited to Non-Destructive Testing (NDT) and do not require force/torque sensing and control. A notable application of a robot used for more than just inspection is the SADIE series robot, pictured in Figure 2.6. This robot carries a specially designed grinding package so that it may remove ladder brackets obscuring its view of welds that it is meant to inspect [45]. While force sensing is available through a differential pressure sensor, the main mechanism that makes grinding possible is a specially designed mechanical (passive) compliance. There is also another interesting application of force sensing and control on this robot. The robot has to be able to climb in an upside down position, and in this case force sensing is used to



ensure that its legs do not push off with too much force which could cause it to push itself off the surface that it clings to [45].

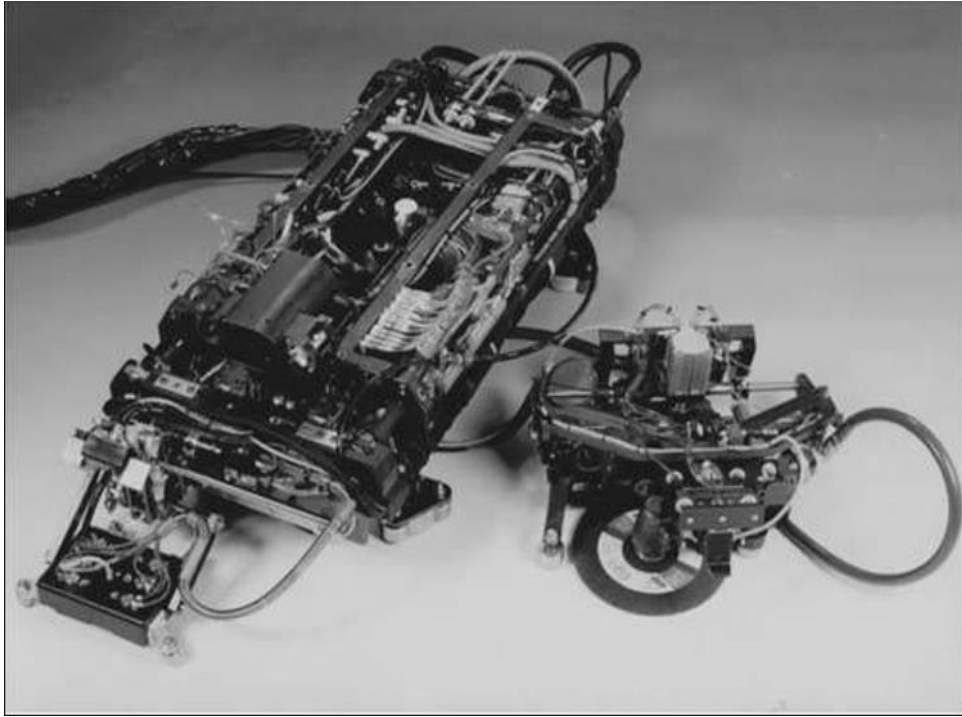


Figure 2.6 SADIE robot [45]

The Savannah River National Laboratory designed another novel inspection and maintenance robot to remove a section of duct work from a highly radioactive environment. The worm type, pipe crawling robot was successful in using a custom plasma arc torch to cut the duct [46]. Note that this is not an operation that requires contact control capabilities.

Decommissioning is required when a nuclear power plant has reached the end of its life cycle. As some components of the power plant are highly radiated, it is useful to employ robots to handle some of the decommissioning. Most robotics currently used in decommissioning are tele-operated with no autonomy or programmed motion. These robots typically use master/slave manipulation or remote control [44].

Inspection and decommissioning are two of the most successful applications of robotics in the nuclear industry, but they are also two that require little physical interaction with the environment (in the case of inspection), and little care for the amount of force applied (in the case of demolition for decommissioning). More finesse might be required when making repairs on inspected systems, or sifting through the rubble of the decommissioned power plant to sort the waste left over. Yet, there is not much mention of these actions in the 2011 review of robots in the nuclear industry [44]. It is also important to note that, according to [45], the costs of the climbing inspection robots is too high for most industries because they are tailor made for each application and, as of 2006 when the article was written, they are only used when there is no alternative.

Another application of robotics in the nuclear industry is handling nuclear materials within a glovebox. This type of application is exemplified by the automation of the Advanced Recovery and Integrated Extraction System (ARIES) at Los Alamos National Laboratory (LANL). The ARIES line converts retired plutonium pits into oxides for use to make mixed oxide fuel, packages the oxides, decontaminates the packages, and surveys the packages. The extent of automation in the ARIES glovebox involves a conveyor system, a pair of 3 DOF gantry robots with custom designed EEFs that add additional degrees of freedom, and two 5 DOF industrial Fanuc robot arms. Figure 2.7 shows one of the ARIES gloveboxes. Although this system is fairly complicated and some parts are completely autonomous, there is limited sensor feedback which restricts automation to mostly pre-planned pick and place type of commands. “The lack of sensor feedback is a significant limitation in RIPS as well as the other automation systems currently deployed but may be remedied with the next generation of systems” [43]. Although the lack of extensive integration of sensors such as force/torque sensors is due to communication limitations in

the case of ARIES, it follows the trend that force/torque sensing is scarcely used in nuclear applications.



Figure 2.7 Fanuc arms in an ARIES glovebox [43]

From the literature, robotic systems have not been utilized for tasks requiring force control or contact monitoring. Yet, there is an extensive set of tasks that could be automated if contact forces could be controlled or monitored. Some examples include:

- Manufacturing assembly/disassembly
- Material decontamination
- Material reduction
- Material packaging
- Grasp verification/Contact inspection

One issue to consider is the use of sensors to collect force data in hazardous environments. However, force data has been collected as a part of several systems that have been deployed across the DOE complex. There is the Spherical Vessel Decontamination (SVD) [47], the tele-operated haptic feedback systems at ORNL [48], and the few listed previously in this review.

## **2.5 SUMMARY OF LITERATURE REVIEW**

Researchers developed the fundamental ideas used in modern applications of compliant control in the early 1980s. Most force and compliant control, even in modern systems, revolves around force/position hybrid control, impedance control, or some combination of the two. Although the initial theories rely highly on setting up the task space and knowing a precise model of the robot dynamics and the environment, later research expanded compliant controllers allowing them to learn stiffness and function in uncertain environments. Rapid advances in computing technology led to a resurgence in the fields of control and artificial intelligence. The robotic community used these advances and integrated them into the ideology of compliant control.

The nuclear industry has limited its use of compliant control techniques and focused narrowly on a few applications. Most applications in the nuclear industry employ custom designed robots and/or custom designed EEFs that allow for a lack of advanced contact control techniques. Robotics in the nuclear industry lags the state of the art and has room for improvement, especially in the field of active compliant control. Roboticist can only accomplish improvement by making industrial robot technologies safer and more reliable and by making advanced robotics methods universal and easier to access. Also, the cost of designing application specific platforms for each task restricts the current use of robotics. Roboticists can make it feasible to use robotics for more applications by making robotic

platforms that are able to perform tasks across multiple applications by generalizing control and allowing for easier automation.

## **Chapter 3**

### **Generalized Contact Control Framework Design**

To overcome the barriers that limit the use of contact tasks in procedures for industrial robots, a generalized contact control framework, GCCF, was designed and implemented. GCCF is designed to control the movement of a robotic EEF while it is in contact – or about to come into contact – with the environment. In this chapter, we will discuss the design of the framework. Since the framework is hardware agnostic, we will not discuss design of the hardware integration or the implementation into code until later chapters.

#### **3.1 DESIGN OF THE GENERALIZED CONTACT CONTROL FRAMEWORK**

GCCF uses intuitive, simple rules to allow a programmer to easily execute contact or co-robotic tasks. It is built to be modular and extensible. Modularity allows for easy access to the control framework’s features, and extensibility allows for improvement as unforeseen challenges are encountered.

The role of GCCF is to accept force and torque data from a sensor or sensors on the robot and determine the appropriate velocity commands that reduce undesired contact forces on the robot and the manipulated object. GCCF executes motion by sending EEF velocity commands to an industrial robot controller. Velocity commands were chosen as the output because velocity control is robust to variations in the control signal and more effectively reduces collision forces than position control [49]. They were also chosen under the assumption that the interface to an industrial robot controller would not allow low level control of torques. Figure 3.1 shows the interfaces to GCCF. GCCF is hardware agnostic so that it may be used on many hardware platforms. It should be usable with any standard

industrial robot controller and 6-axis force/torque (FT) sensor. The implementation chapter will further discuss the ability to interface with a wide variety of hardware.

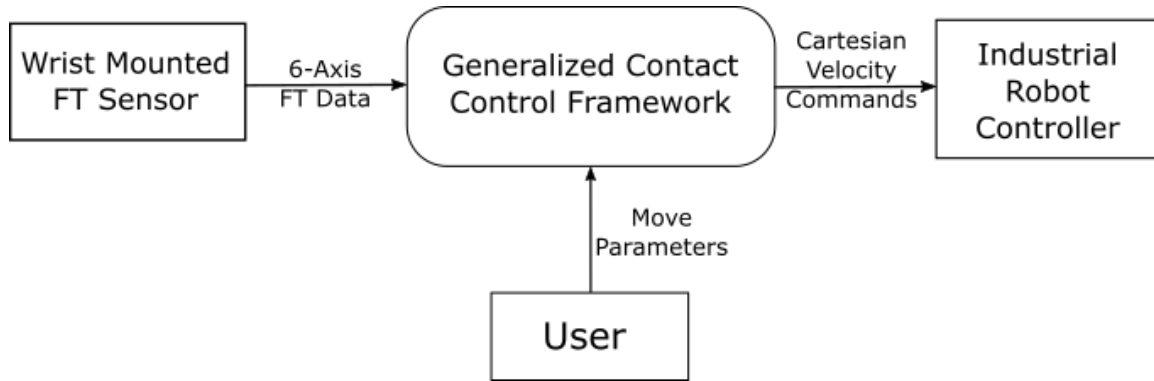


Figure 3.1 Design interfaces

### 3.1.1 Control Dimensions and Reference Frames

The most powerful concept of the GCCF is that a user can control each dimension of a user defined reference frame independently. This allows a user to break down a complicated contact task into simpler parts and control each part separately. The user can pick a reference frame in a manner that is similar to Craig and Raibert's hybrid position/force control approach [24]. By selecting a Cartesian reference frame that is orthogonal to natural constraints, the user may break up the contact task problem into a set of force control problems and a set of position control problems. The user is then able to perform all control within this convenient reference frame while GCCF handles transformations of all data and control between frames. Figure 3.2 shows the transformation frames that the framework is designed to handle. The FT frame is the reference frame of the force/torque sensor. GCCF must convert force/torque data to the task frame so that it can make calculations in the user defined reference frame. Before sending velocity commands, GCCF converts them into the robot control frame so that the robot controller can understand them. The user specifies all reference frames shown by any

transformation from the global reference frame or any link of the robot. This allows the user to have lot of flexibility in the specification of the task frame, and also allows the FT frame to update position and orientation as the FT sensor moves with the robot motion.

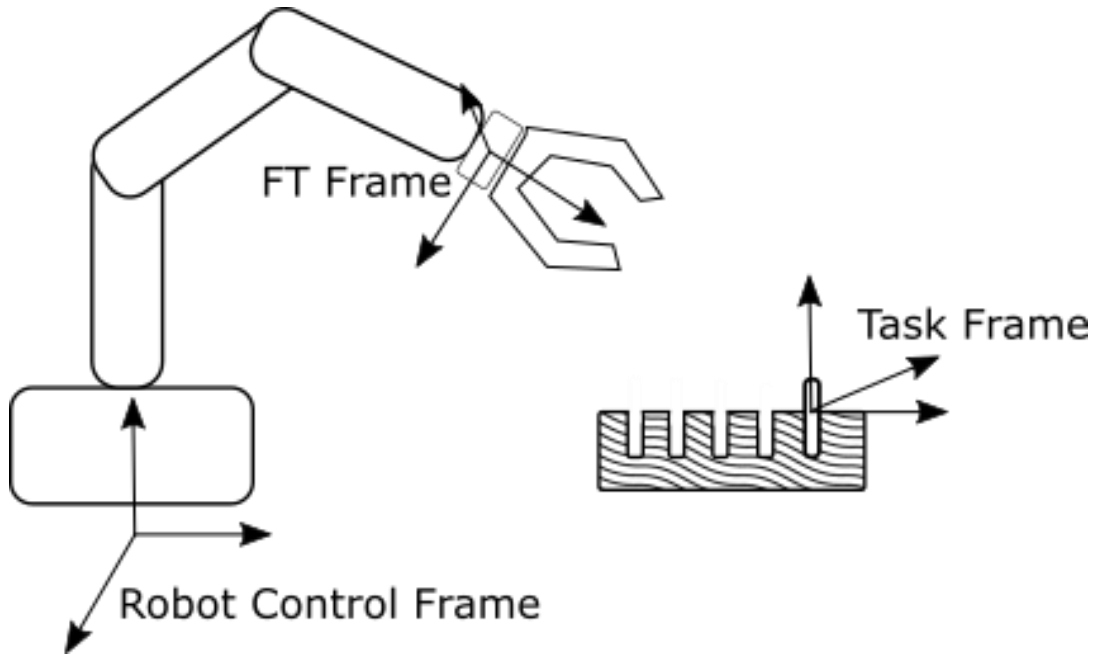


Figure 3.2 Managed reference frames. The FT frame is the frame that the force/torque data is read in. The robot control frame is the frame that the framework sends robot commands in. The task frame is the frame that the user specifies control task in.

As discussed in Chapter 2, the literature has questioned the validity of the hybrid force/position controller. There are two differences between the generalized contact control framework and Raibert and Craig's approach [24] that allow GCCF to avoid the issues found [25] [26] [27] in the hybrid force/position controller. One difference is that there is no explicit force control or explicit position control in the framework. Each dimension either uses one of the impedance based control laws defined in section 3.1.2 or is not controlled at all. In some cases, the user might decide to control a dimension that could use simple position control with one of the control laws. This can help with two problems. One



problem is that the user might not be able to precisely line up the reference frame orthogonal to the natural constraints, or the constraints may not be divisible by a standard set of orthogonal axes. In the case of Raibert and Craig's hybrid force/position controller, some component of the contact forces that should be controlled on an explicit force control axes may actually fall to the direction controlled by explicit position control. The component of contact force that is being mismanaged by the incorrect frame specification will stress the robot improperly. The other problem is that the user might not have precise knowledge of the environment and therefore may not know if an axis should be force or position controlled. In either case, a user of GCCF could use an impedance control law on what would be an explicitly position controlled axis in a hybrid force/position controller to attempt to maintain position control while still adapting to unwanted contact forces. Section 3.1.2 discusses use of the impedance control laws for such purposes.

The second difference from the hybrid force/position controller is that GCCF does not transform the movement with respect to the Cartesian reference frame directly to commanded movement in joint space. It only transforms from the desired Cartesian control frame to the robot controller's control frame and sends a Cartesian velocity command to the controller. The industrial robot controller then maps these commands to the individual joint controllers. We have assumed that the industrial robot controller is stable under expected input conditions. This does not necessarily mean that any possible combination of inputs to the industrial robot controller that GCCF can generate would lead to stable behavior. It is beyond the scope of this thesis to prove the stability of the coupled system consisting of GCCF and any industrial robot controller for all inputs. It is up to the user to verify that the output generated by GCCF does not disagree with the selected controller's expected input quality. We specifically heed caution for tasks that involve rapid vibration which would lead to rapidly oscillating velocity commands.

### 3.1.2 Control Laws

GCCF makes available three control laws for the user. While all the laws could be simplified into one law with many options, we believe that this categorization allows the user to intuitively understand the purpose of each law and when to apply them. It also allows for a simpler interface requiring less parameters to be decided on by the user. The control laws are formulated in equations 3.1, 3.2, and 3.3.

$$v_{c, \text{follower}} = \frac{1}{b}F \quad (3.1)$$

$$v_{c, \text{spring}} = \frac{1}{b}F - k(d_{\text{travel}} + d_{\text{offset}}) \quad (3.2)$$

$$v_{c, \text{compliant move}} = v_{\text{desired}} + \frac{1}{b}F \quad (3.3)$$

where  $F$  is the force (or torque) sensed in the dimension being controlled,  $d_{\text{travel}}$  is the distance the EEF has moved in the control dimension since the start of the move, and  $v_c$  is the commanded velocity sent to the robot controller. The variables  $k$ ,  $b$ ,  $v_{\text{desired}}$ , and  $d_{\text{offset}}$  are user supplied constants that determine the behavior of the control law. Note that these laws were designed to be simple and user friendly, but GCCF could be easily modified in the future to allow for more complicated laws, e.g. ones that track precise forces or adapt for unknown stiffness. The surrounding framework and ability to control different laws in different dimensions would still be intact and useful. For now, the design will not be used to track precise forces, rather it will be used to allow for safe and easy robot control to contact the environment and manipulate environmental objects.

Equation 3.1 is the follower equation. This is the simplest of the control laws. If an axis is controlled in this manner, it will react to forces by moving in the direction that the force is pushing it. The use of this rule is exemplified by the peg in the hole problem seen in Figure 3.3. If there is contact with the walls of the hole as the peg moves into the hole, the peg should move slightly away from that wall. Ideally, there should be zero net force

in all directions normal to the wall. In this case, the desired position, i.e. the center of the hole, comes from the force profile. No reference position is necessary.

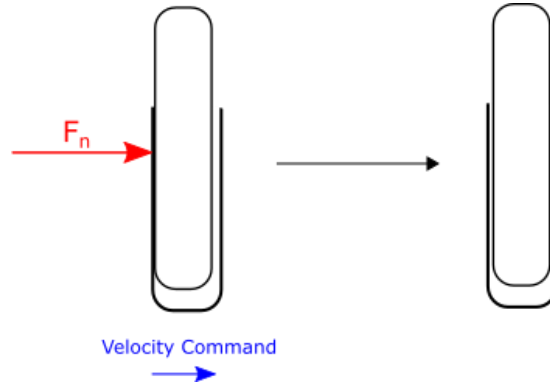


Figure 3.3 Peg in hole problem using follower law

The only parameter that the follower equation requires is  $b$ . The  $b$  parameter linearly maps the force sensed to a velocity command as a scaling factor. The greater the force, the greater the commanded velocity. In the case of the peg in the hole problem, a user would tune this parameter so that the commanded velocity due to the contact force with the wall is not so large that the peg moves back and forth between the two walls, but is not so small that it does not effectively move the peg off of the wall.

Equation 3.2 is the spring equation. This equation models the idea of a spring by including a resistance to motion away from a desired, or unstretched, position. Note that the user sets the reference position via the offset variable. The user may specify the offset position beyond an obstruction so that the robot will apply a force on the obstruction. In this case, it is as if the obstruction is compressing the virtual spring to be shorter than its unstretched length. One use of this equation is surface following (assuming a constant force on the surface is not required). This can be seen in Figure 3.4. When the control law is initialized (in the first state of the figure, Move Start), the arm is just touching the surface.

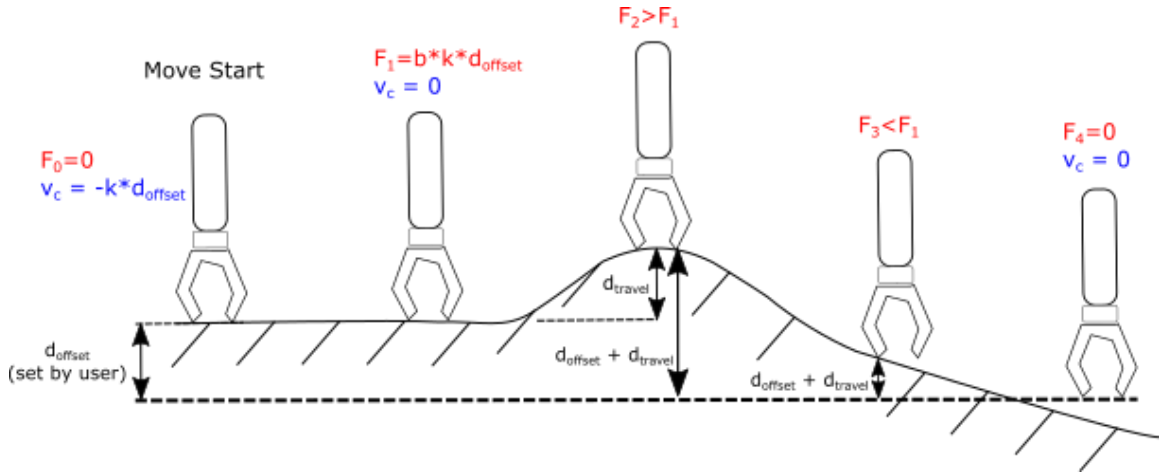


Figure 3.4 Surface follower problem using spring law

While the surface is flat, the arm starts to apply a force, equal to  $F_1$  in the figure, since the resting length, seen by  $d_{offset}$  in the figure, of the virtual spring is set to be farther than the robot can reach. As the surface changes to be higher, the virtual spring is compressed farther from its resting position and the force applied to the surface is larger. Likewise, as the surface lowers closer to the resting length of the spring, the spring is allowed to decompress and the force becomes lower. If the surface falls below the point where the virtual spring is at its resting length, the robot's height remains constant at that point.

Another motivation for a user to use this law was mentioned previously as one of the reasons GCCF does not succumb to the same pitfalls as a typical hybrid position/force controller. A user can apply this law if movement is not desired in a dimension. If the stiffness is sufficiently large enough, the robot will not move much unless a large contact force is encountered. As discussed before, this can be a good idea if the environment is not known precisely.

Equation 3.3 is the compliant move equation. In this equation, a desired movement velocity is opposed by the external force at the EEF. A user can apply this equation to

attempt to follow a velocity commanded trajectory while avoiding excessive force on the EEF or to move until contact. Often in contact tasks, the robot is unaware of the precise position and orientation of the environment. In order to find an appropriate start position for the contact task, the EEF must move from free space into contact with an object. This move is visualized in Figure 3.5.

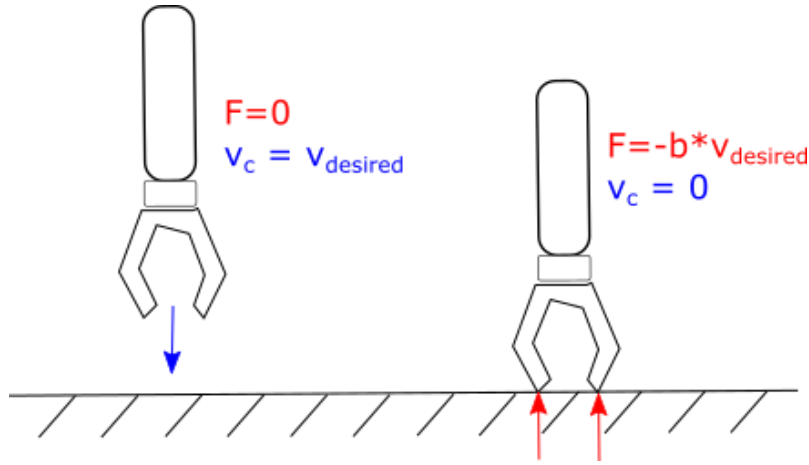


Figure 3.5 Guarded move problem using compliant move law

Using this compliant move equation, a user can command the EEF to move to a location beyond the object's estimated location and it will stop moving once in contact with the object. The *stall force*, or the force on the EEF when contact has caused it to stop, can be calculated by setting the commanded velocity in equation 3.3 to zero, as seen in equation 3.4.

$$v_c = 0: F_{stall} = -b * v_{desired} \quad (3.4)$$

As stated previously, the control laws are a synthesis of existing research defining the concepts of stiffness control, damping control, and impedance control. The origins of these concepts can be found in the literature review section. The laws are a form of impedance control, proposed by Hogan [30], which is a generalization of stiffness and

damping control. Like damping control, the laws use velocity modifications to reduce contact forces, and like stiffness control, the spring law resists movement away from some nominal position. They are not completely new ideas, but here they are presented in a form that would be easy for the operator to understand and use.

The equations were chosen to be intuitive for the user's understanding and analogous to physical systems of which the user has knowledge. As stated previously, each axis could employ a single universal control law, but the result would be less intuitive to all but developers with extensive controls experience. The objective here is to eliminate the need for such experience when developing robotic systems to address contact applications.

Note that a user can easily understand all of the parameters required for the control laws in terms of the force of a spring or an inertial mass. In all cases,  $b$  determines how much the sensed force or torque affects the motion of the robot. This is analogous to the inverse of the EEF's simulated inertia. A small  $b$  means that the force applied easily impacts the EEF's motion. A large  $b$ , on the other hand, simulates a large mass that requires a large force to affect its motion. In the spring equation,  $k$  is analogous to the spring constant in the familiar linear spring equation from Hook's Law,  $F = kx$ . In our control environment, a command to move in a direction applies a force on an object that is in contact with the EEF. The  $k$  variable determines the impact of the displacement from the spring's unstretched position on the movement of the EEF. These relations to physical concepts make the tuning of the control laws more intuitive for the experienced, but not expert user.

### **3.1.3 End Conditions**

So far, the design section has limited discussion to how the user will set up a move by defining reference frames and control laws, and how GCCF will execute the move by calculating appropriate velocity commands to send to the robot controller, but the section has not discussed what ends a commanded motion. In the design of GCCF, the end of a move is defined by the user and can happen in multiple ways. Specifically, two end conditions are directly built into the design: maximum force/torque exceeded, and maximum displacement exceeded. These two conditions are explicitly built in because they are also safety features for the framework. The specification of every move requires the user to input a maximum force/torque and a maximum displacement so that the robot will not apply excessive force or move out of the bounds of the task. However, these conditions can also be designed end conditions rather than safety features. For example, if the user requires a robot to move until contact with a surface and then continue slowly to apply some force on that surface, the user would want the move to end when the intended applied force is reached. Or a user might want to move a specified distance in a prescribed direction or allow no more than a specified EEF deflection and therefore end the move at maximum displacement.

The user accomplishes all other possible end conditions through the asynchronous move design feature. The asynchronous move is a feature that gives the user the ability to execute moves asynchronously and then stop them externally. This allows for any end condition that the user would like, as they can command a move and then check their own end condition for an appropriate stopping point.

### **3.1.4 Examples of Setting up the Control Framework on Multiple Axes**

GCCF is most easily understood through example. Two examples of contact tasks that we believe could be accomplished through the use of GCCF follows. The first example

illustrates how a user can account for uncertainty in orientation, not just position. The second example illustrates how a user can perform a task that involves control on multiple dimensions.

The first example, shown in Figure 3.6, is another guarded move. This time, the surface that the robot is trying to make contact with is not orthogonal to the palm normal of the gripper. In this case, the  $y$  dimension, shown by the Cartesian reference frame in the figure, behaves as a compliant move as it did previously. Due to the uncertain orientation of the surface, the user should also add a follower law to control the rotational  $z$  dimension. Initially, there is no torque on the robot and the EEF will move downward until contact as it did with the previous guarded move. Once the left finger of the gripper makes contact with the table, the contact force between the table and the gripper results in a torque on the robot. The follower law on the rotational  $z$  axis converts this torque into a rotational velocity. As the robot continues to attempt to move in the negative  $y$  dimension, it will also start to rotate around the  $z$  dimension until the right finger also makes contact. Once the orientation of the EEF is orthogonal to the surface, the two contact forces will balance each other out to apply zero net torque on the robot arm and rotational motion will cease. Note that this example is only in 2-D for demonstrative purposes, but the 3-D version would also work. In the 3-D case, the user would also add follower to the rotational  $x$  axis to allow for a slant in the  $z$  direction.



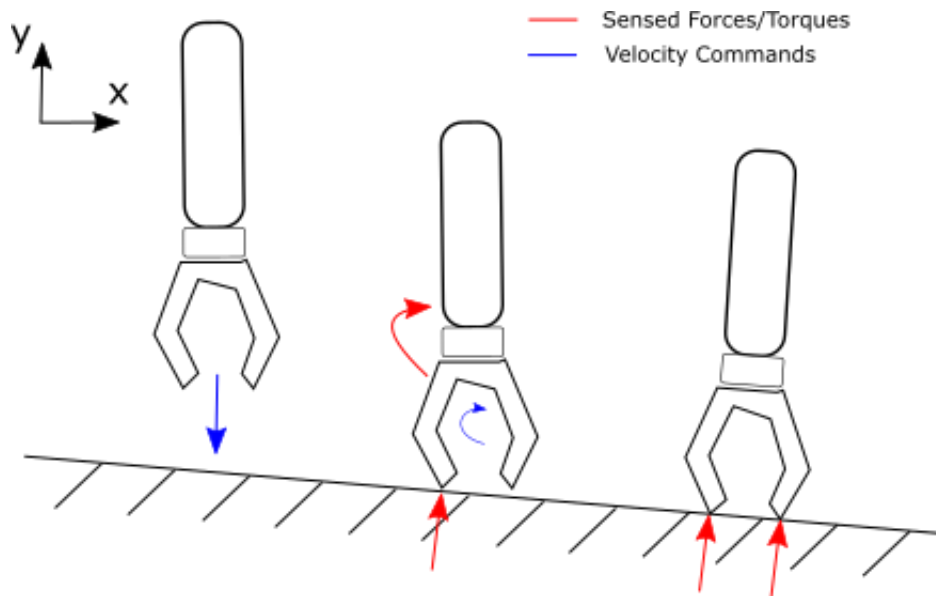


Figure 3.6 Using the control framework with uncertain orientation

The second example, shown in Figure 3.7, illustrates screwing a lid on a container. The figure also shows a user defined reference frame for each type of control law implemented in the move. To set up the control framework, the user needs to think of the artificial and natural constraints discussed previously. To screw on the lid, the robot needs to attempt to move downward (negative  $z$  dimension) and twist the lid in the threaded direction (positive rotational  $z$  dimension). To do this, the user would introduce two artificial constraints by setting compliant moves on the linear  $z$  dimension and the rotational  $z$  dimension. While screwing on the lid, the user might also want the framework to adjust for orientation misalignment. To accomplish this, the user could set spring laws for the rotational  $x$  and  $y$  dimensions. This allows the EEF to rotate if it senses torque pushing away from the vertical orientation, but would not allow the orientation to drift too far from the initial guess of where it should be. The user could also set the  $x$  and  $y$  dimensions as followers. These laws account for a small amount of misalignment in the position of the

container in the  $x$  and  $y$  dimensions. As the lid comes down on the container, contact with the lip introduces contact forces with some component pointing towards the center of the container. The follower law converts these forces into velocity to align the lid. Note that by applying the follower laws, the user assumes that the lid has been lined up well enough that the inside of the lip will contact with the container. If this is not a valid assumption, the user must employ some other method such as vision monitoring or another contact task, e.g. a guarded move, to determine the position of the container more accurately.

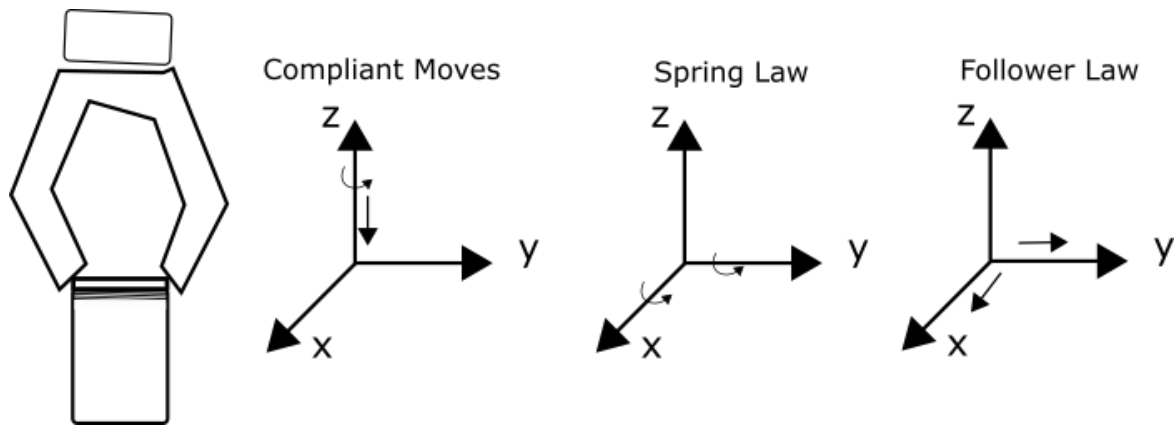


Figure 3.7 Screwing on a lid with the generalized contact control framework

The desired end conditions of this move could be a number of things, depending on what the operator would like. For example, an end condition could be maximum travel in the  $z$  direction if the operator knows how low the threads go on the container. It could also be maximum travel on the  $z$  rotational dimension if the operator knows how many turns would give the desired seal. More likely, the operator would set a maximum torque in the rotational  $z$  dimension to ensure that the lid is tight enough. There would also be end conditions set that ensure unsafe forces and torques are not reached.

### 3.2 DESIGN SUMMARY

The GCCF design allows a user to accomplish a large variety of contact tasks using a simple, intuitive interface. It accomplishes this by splitting the control of EEF motion into separate control of its velocities in each linear and rotational dimension. The control of each dimension can be chosen from one of three intuitive control laws. Each control law design has a physical interpretation and only a few parameters so that a non-expert user can use them to set up a control problem.

It is important to note that the abilities of the framework are limited by the inferences that can be taken from the sensed information at the wrist force/torque sensor. If, for example, the lid of the container in the previous example were not lined up well enough, the contact forces caused by the collision of the lid and the container would not give enough information to deduce the direction of the misalignment. This limitation can sometimes be mitigated by extra contact moves that eliminate some uncertainty in the task or using vision monitoring to deduce approximate locations.

Although the initial design is meant to be simple enough to be understood by the non-expert user, the design leaves room for expansion of more advanced control laws when the need for them is realized. The addition of more control laws would not compromise the benefits of the configurable design. Another limitation of the design might be that it only outputs EEF velocity commands. In the future, GCCF could be expanded to additional command capabilities. Also left to future work is the use or creation of a formalism for the task space itself. Such would be necessary to properly map the capabilities of the controller to a well-defined set of task or formally defined subset of the entire task space.

## **Chapter 4**

### **Implementation**

The implementation of GCCF exemplifies the desired design traits of the framework discussed in Chapter 3. GCCF was designed to be intuitive, modular, and extensible and so the implementation is also intuitive, modular, and extensible. This was accomplished with the use of object oriented programming, and incorporation into ROS (Robot Operating System) [13]. In this section, we will describe the tools used for implementation of the GCCF's design, and how the choices that were made in the implementation satisfy our design goals.

#### **4.1 REQUIRED HARDWARE**

Although the GCCF is hardware agnostic, it requires multiple pieces of hardware to run on and interface with. The code is written in C++ and embedded within ROS, as will be discussed later. This requires that GCCF run on a Linux computer that has all appropriate system requirements to run ROS. The robot being controlled by GCCF is assumed to be an industrial robot, but can be any serial manipulator with pre-built control capabilities. GCCF controls the robot with EEF velocity commands through ROS topics (to be discussed later). If the robot is not able to take commands in this manner, the user will need to write the appropriate drivers to incorporate the robot into the ROS infrastructure and, if necessary, convert the velocity jogging commands into commands that the robot's controller can accept. In order to make the contact control calculations, GCCF needs access to a 6-axis force/torque (FT) sensor. The FT sensor should be wrist mounted on the robot so that it may sense the forces and torques applied to the robot's EEF. Wrist mounted means that the sensor is attached to the last link of the robot and supports the weight of the EEF entirely. These are all the necessary hardware components used to

run GCCF. It is important to remember that GCCF is hardware agnostic, and thus it may be necessary to implement hardware drivers or other conversion code; however, the drivers for many robots and compatible sensors already exist in ROS.

## **4.2 ROS**

Before the details of the C++ implementation of the framework are discussed, it is important to understand ROS, Robot Operating System. ROS was the chosen framework to contain the GCCF. While it is called an operating system, ROS is really a collection of libraries that allow for easy integration of robot operating code and unification of conventions for controlling that code. ROS is widely used in the robotics research community and many useful tools are already published as ROS packages (a group of ROS code designed to interface with other ROS code). A user of ROS can easily integrate these packages with new robotic code. In this section, we will discuss the details of ROS plumbing, and the existing ROS packages that were incorporated into GCCF.

### **4.2.1 ROS Plumbing**

The building blocks of the ROS environment are nodes and messages. There is little about the *plumbing* of ROS that makes it specific to robotics. ROS is simply a set of tools that allow a user to structure code in a way that makes it easy for another user to incorporate into their own ROS code.

#### ***4.2.1.1 ROS Nodes and Messages***

The ROS environment is built around the idea of independent nodes that connect to each other through peer to peer messaging services. By using nodes and messages, ROS is made to be modular. Although the setup is very different than object oriented programming, the concept can be understood in a similar fashion. In object oriented programming, objects are supposed to be created in a manner so that a program that

instantiates the object does not need to know about the inner workings of the object. The program only needs to access specific public functions, or interface functions, that allow the higher level program to use the object in the way that it was intended. In this same manner, any ROS node does not need to know about the inner workings of any other ROS node. All a ROS node needs to work with another ROS node is a list of the messages that are output by the node and a list of messages that are expected by the node. A text file for each message type defines each available message for a package or ROS node. These text files, along with built in ROS functions, allow users to see the required structure of the information requested by the node (or published by the node) for other nodes to use.

There are three ways to use the ROS messaging structure: topics, services, and actions. A ROS node publishes messages to topics so that any other ROS node subscribing to the topic can read the message. This allows nodes to share data with other nodes. An example of the use of a topic is the force/torque data topic. A ROS force sensor driver node, built to make a specific force sensor ROS compatible, might read in data from the force sensor with the protocol required of the sensor. The driver node outputs the data in a general form as a ROS wrench message to a force data topic so that any other node can read the data at any time while the driver node is running.

Services allow for one-time, two-way communication. Every service consists of a request message and a response message. Services are like public access methods of a class. To call a service, one node passes information (the request) to another node and then waits for information to come back (the response) that signals that the communication was received and acted upon. The third type of messaging structure, actions, are a structured combination of a service and published messages that allows a node to ask for an action to be completed, and then publishes the status of that action while it is executed.

#### **4.2.1.2 ROS Core**

ROS core is the background code that must be running for ROS nodes and messages to work. When a ROS node is started, it registers itself with ROS core so that other nodes can connect to it [13]. ROS core establishes the appropriate connections so that messages may be received and transmitted by ROS nodes. To do this, ROS core also keeps track of node names, message topics, and parameters so that they can be easily queried by a ROS user. ROS core also hosts the parameter server which is a list of configuration variables that can be set and read by any ROS node.

#### **4.2.1.3 ROS Packages**

A ROS package is a node, or a set of nodes, that a designer has developed and organized in a standalone fashion along with the documentation required to interface with the package. A ROS user can upload or access packages through the ROS wiki page [50]. With a simple Linux install command, a ROS user can gain access to a new ROS package developed by another ROS user. A single ROS package may be just one node, or it may be a set of nodes, that communicate internally and with each other over the ROS messaging structure. The ROS *plumbing* is visualized in Figure 4.1. The visualization shows the inner workings of a package on the left that is made up of 3 nodes communicating with each other over 2 message topics and a service call. Although the package on the left knows nothing about the other package on the right, it is able to communicate to it by advertising a topic that sends out messages that the other package can read. In fact, these packages don't even need to be written in the same language to communicate. The ability of the messaging structure to support multiple languages is another strong point of ROS. For more information about ROS infrastructure, see [13].

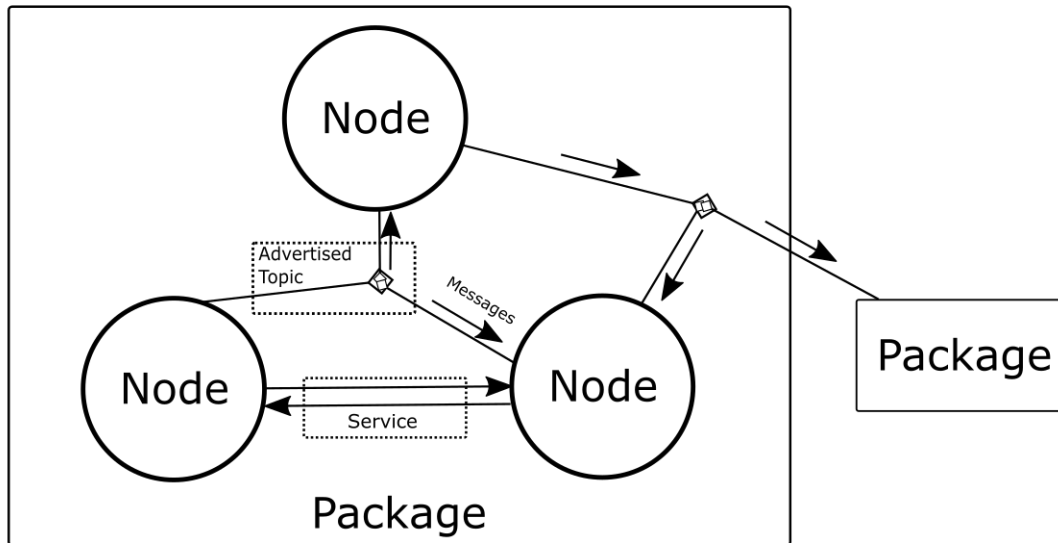


Figure 4.1 ROS plumbing

#### 4.2.2 TF: ROS Package for Managing Cartesian Coordinate Frames

Along with the prebuilt infrastructure, the most useful part of ROS is the community of packages that users have built that accomplish many tasks that are needed in any robotic application. One package heavily used in GCCF is TF. TF is a ROS package that keeps track of user defined coordinate reference frames and provides easy to use functions that transform many types of data from description in one frame to another. GCCF uses the TF package to transform the force/torque data from the raw input frame to the user defined control frame. This would be especially tricky without TF since the FT sensor is attached to the robot and is constantly moving with the last robot link. The motion planning package, to be discussed in section 4.2.3, constantly updates TF frames that describe the location of the robot links. The relationship between the last robot link and the FT sensor therefore yields constant TF updates to match the real location of the FT sensor. GCCF also uses TF to transform the robot commands into the appropriate frame that is



required by the robot controller for velocity commands. The names of these frames can be configured at start up through a configuration file that is read into the ROS parameter server.

#### **4.2.3 MoveIt!: Motion Planning and Robot Description**

Motion planning and kinematics are accomplished through a ROS package called MoveIt!. Although GCCF does not use motion planning to accomplish its main goals, it is important that it can easily move from one contact task space to another. Any demonstration implementing GCCF for contact tasks will have to control the robot via MoveIt! to position the robot in the correct position to start each task. MoveIt! allows a user to command a ROS integrated robot simply by specifying a goal position in the form of a hand pose goal or a joint set goal. MoveIt! communicates directly to the robot driver package via ROS messaging to control the robot when commanded. GCCF does, however, interface directly with MoveIt! to make use of MoveIt!'s kinematic capabilities. MoveIt! loads a description of the robot's link lengths and joint connections when it starts and uses this description and live data coming from the robot driver to keep track of robot kinematics at all times. Through MoveIt!, GCCF can determine the current pose and joint angles of the robot. GCCF uses this information to determine the displacement from the start position so that it may implement its impedance control laws.

When commanded to move the robot, MoveIt! not only generates possible planned trajectories, but it also evaluates these plans based on user constraints and collisions with known environment objects. To generate trajectory plans, MoveIt! uses the Open Motion Planning Library (OMPL) which is an open source motion planning library that implements motion planners. The user can pick any of the available motion planners which are mostly randomized motion planners [8]. MoveIt! also implements collision checking

via the Flexible Collision Library, FCL [8], and allows the user to add collision objects to the modeled world. MoveIt! keeps track of collision objects to avoid colliding with them. In the setup of any demonstration using GCCF, the user should add conservatively sized collision objects to MoveIt!’s planning scene around any contact control areas to avoid contact while GCCF control laws are not being used. After a MoveIt! move, GCCF can be used to safely move closer to the contact task area and to perform the contact task manipulation.

### 4.3 C++ IMPLEMENTATION

GCCF is written in C++. The user interface to the code is through a set of public C++ functions in the contact control class. Although we will later show a demonstration of a graphical user interface (GUI) wrapped around the framework, it is primarily envisioned that task parameters will not be adjusted by the user during a task. Thus parameters can be fixed at compile time, imported from a text file, or determined algorithmically. In a later demonstration, a GUI does illustrate how easily the framework can be configured by allowing user controls during motion. The framework, without the GUI, is meant to be used to assist a worker that programs robotic demonstrations and/or procedures.

Figure 4.2 shows a class diagram for the main pieces of GCCF. The interface to the framework is through the `ContactControl` class. A procedure or demonstration function will instantiate this C++ class. There is currently no ROS messaging wrapper around this class to allow for multi-language use, but one could be added in the future. The `ContactDirection` class is the class that does most of the work in applying the control laws. `ContactControl` instantiates it as an array of 6 instances, one for each dimension of control. There are also several enumeration structures that allow for named directions, control laws, and enumeration of the exact reason for a contact move to be ended.

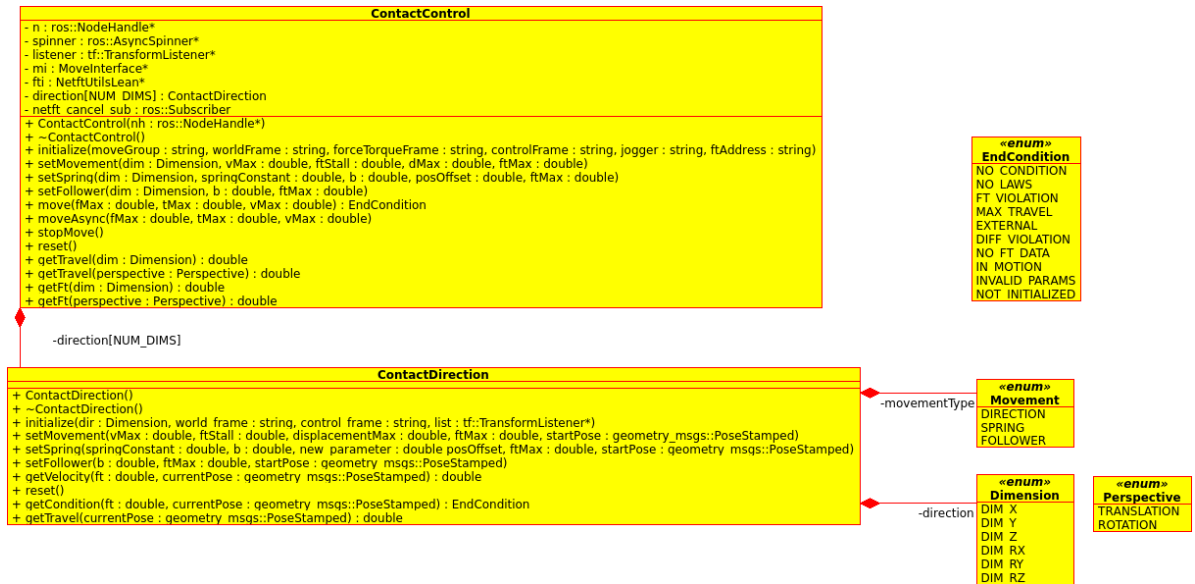


Figure 4.2 Class diagram of generalized contact control framework

Once instantiated, the user must initialize the `ContactControl` class to configure the user defined TF reference frames that should be defined in the robot description that was loaded by MoveIt!. The initialize function also configures the type of jogging message required to communicate with the robot driver and the IP address of the FT sensor. `ContactControl` class instantiates several objects that allow interfacing with the tools we have previously discussed which include:

- ROS node handle, asynchronous spinner, and subscriber
- TF transform listener
- Move interface
- NetFT Utils (lean version)

The ROS node handle and spinner are required to send and receive ROS messages. The TF transform listener is required to receive transformation information from TF. `MoveInterface` and `NetftUtilsLean` are two in house ROS packages that assist

the interface to MoveIt! and the FT sensor. These two packages will be discussed in section 4.3.3.

Figure 4.3 shows the flow of a typical contact task procedure. To perform a contact task, a user has to first set up control on each dimension that is to be controlled. It is not necessary for the user to control every dimension. The user might desire to leave a dimension stiff if there is no need for compliance in that direction. The example in Figure 4.3 only sets control for the linear  $x$ , linear  $z$ , and rotational  $z$  dimensions. To set up a dimension, the user picks a control law (follower, spring, or compliant move as discussed in the design chapter) and calls the appropriate access function (`setFollower()`, `setSpring()`, `setMovement()`, respectively). With any of the three functions, the user tells the `ContactControl` class which dimension to set as the specified control law, all of the necessary tuning parameters for the law, and the max force allowed in the dimension that the control law is being specified.

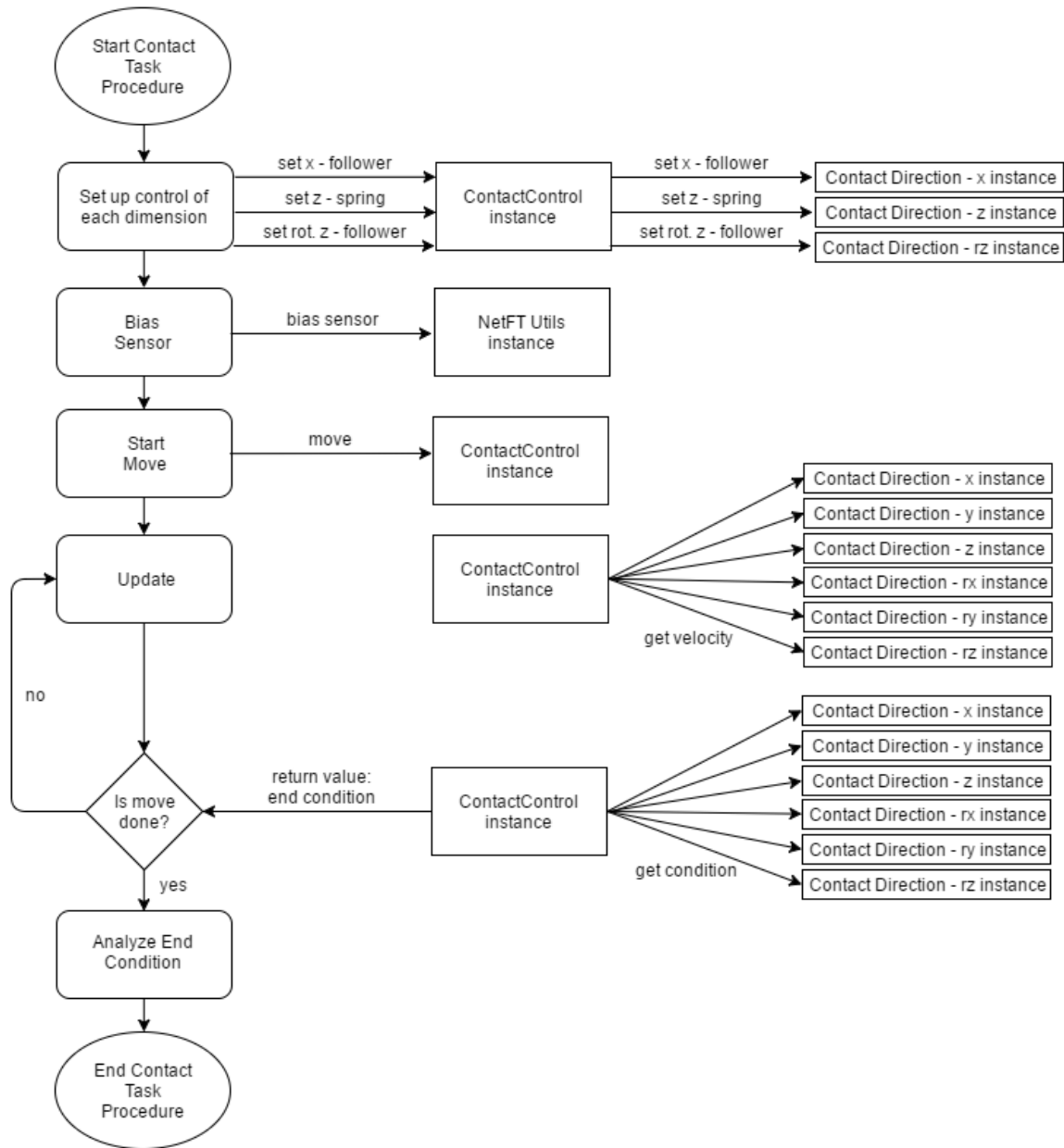


Figure 4.3 Typical flow of contact task procedure

Once the control task is set up by calling the appropriate setter functions, the user executes the move by calling either the `move()` or `moveAsync()` function. The `move()` function will block until the execution is finished and return the reason that the

move has finished. The `moveAsync()` function uses C++11 built in threading functionality to execute without blocking. C++11 is the version of C++ that first implemented this functionality. It is out of the scope of this thesis to explain C++11 asynchronous threading, but tutorials are available [51]. The return variable is of type `std::future<EndCondition>`. The user must store this variable and then call the `get()` function to join the thread and receive the end condition. The asynchronous move will not work if the user does not call `get()` function to join the thread. If the move is not finished when the `get()` function is called, then the `get()` function will block until completion. The `moveAsync()` function may be used to establish an end condition that is not incorporated into GCCF by calling the `stopMove()` function when the end of the move is desired. An example of using the asynchronous move function and stopping the move externally is shown in Appendix A.

The `ContactDirection` class has many of the same methods as the `ContactControl` class. The `ContactControl` class routes commands received by the user to the appropriate contact direction. The `ContactControl` class sends the velocity commands to the robot driver after calling the `getVelocity()` function of each contact direction. The `getVelocity()` function sends the `ContactDirection` instance the sensed force or torque value after the `ContactControl` class makes the appropriate transformation from the force/torque frame. The `getVelocity()` method takes this information and calculates the appropriate velocity command to send to the robot driver and returns it in the same function. This velocity is then received by the `ContactControl` class and transformed into the control frame of which the robot driver accepts commands. This process can be better understood by looking at the flow of Figure 4.3.

### 4.3.2 Safety in the Implementation

Safety is very important when implementing contact tasks. When using a stiff, industrial robot, it is possible to command the robot in a manner that will cause damage to the environment or the robot itself. The main safety features of GCCF are executed through the monitoring of the 6 axis FT sensor since this is the data accessible within the framework.

The most basic safety mechanism GCCF employs is that it will not execute any commands if no data is received by the FT driver. The next level is within the control law itself. Each control law requires that the user input a maximum force or torque for the controlled dimension when the contact task is being set up by the public setter functions discussed previously. Usually this value is useful not only for safety, but for applying the appropriate force to the system for the goal the user is trying to accomplish. If this value is exceeded, the contact move will be stopped and the `move()` or `moveAsync()` function will return an `FT_VIOLATION` end condition. When this value is received, an appropriate code block should decipher whether this end condition was intentional or not. The difference in intention is usually related to the way that the user set up the control dimensions. If the dimension that stopped due to excessive force or torque was meant to apply a force or torque, then this is probably an intentional FT violation. If the dimension was just meant to maintain a position or adjust for inaccuracies, then this was probably a safety stop.

The next level of safety is the overall maximum force or torque which is a required input in the `move()` or `asyncMove()` function when the move is started. The difference between the previously discussed maximum, is that this maximum is the overall magnitude, not just the one-dimensional value. GCCF sends this max value the force/torque utility package which monitors the biased sensor data directly from the force/torque driver. If this

value is exceeded, the force/torque utility package issues a cancel command via ROS messaging and any package that is monitoring that command knows to cancel what it is doing. For our purposes, this command is monitored by the `ContactControl` class and the `MoveInterface` class that will be discussed in section 4.3.3.2. This level of safety monitoring is useful if the environment applies a force on a control dimension that GCCF is not controlling with one of its available laws, or if the forces on multiple dimensions are totaling to a value that the operator feels is unsafe for robot operation.

It is worth mentioning that these levels of safety are only a piece of the safety architecture that a GCCF user should implement for any robotic procedure. Safety at the Nuclear and Applied Robotics Group (NRG) is executed with the safety architecture shown in Figure 4.4. The operator executes the bottom level of safety shown in the diagram. While prototyping demos or procedures, an operator should always be holding an e-stop and looking for signs of excess stress in case all other levels of safety fail. The industrial robot controller executes the next level of safety. Every industrial controller should shut down the robot when joint torques are excessively high. The next level is torque based collision detection. The robot driver implements this level by monitoring the joint torques in a manner described by Schroeder [47]. This type of collision detection is more sensitive than the collision detection on the industrial controller.



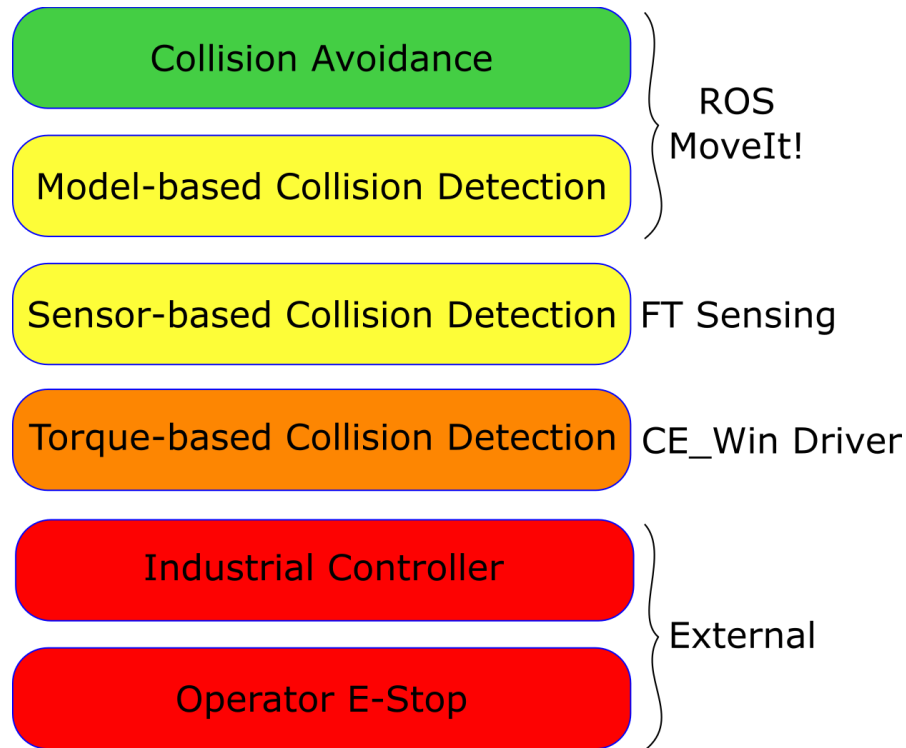


Figure 4.4 NRG Safety Architecture

It is important that multiple levels of safety are implemented when running GCCF. As a consequence of lag in the system and communications, the end condition of the robot after a stop is commanded by the framework will necessarily allow more force or torque to be applied to the robot than was set by the max force and torque variables passed to the framework. Since GCCF is a generalized framework that is built to be used on any system, the amount of excess force or torque cannot be characterized and accounted for and will be different depending on the user's setup. For this reason, it is important for an operator to start developing with some initial built in compliance to the system. This safety compliance could be a foam pad on top of a hard surface or some compliance in the EEF itself. This extra compliance should be used until the nature of this lag and capabilities of the system

are understood by the operator. It is also important to start with very conservative parameters when tuning and work up to the appropriate parameters for the final demonstration or procedure.

### **4.3.3 Other in House Code Supporting the Framework**

Other than the code written directly for contact tasks contained in GCCF, there were a few other packages written for the framework and for other projects at the NRG, that allow for easier access and interfacing with the robot and the force/torque sensor.

#### ***4.3.3.1 NetFT Utilities Package***

The NetFT utilities package manages the raw data coming from the NetFT driver package and publishes it in a more useful form. The NetFT utilities package takes in the raw data from the NetFT driver and applies bias and thresholds to the data before advertising it on a ROS topic. Users can set the bias and threshold values via a ROS service call and update them at any time including during run-time. The package also has the ability to transform the data from the force/torque frame to a tool frame. Users can specify both of these frames in the configuration file loaded at startup. The NetFT utilities package also keeps track of a maximum allowed force and torque and advertises a cancel message when the maximum is exceeded. The package was further modified for GCCF with the addition of a lean version. Users can instantiate the lean version directly to avoid passing the same information too many times as ROS messages. The capabilities of the lean version were reduced to run as fast as possible. This is useful to keep the lag time down for the framework. The lean version also includes a low pass filter which can be turned on at any time to filter the incoming force/torque data. Depending on the force/torque sensor the user chooses, this may or may not be necessary.

#### ***4.3.3.2 Move Interface Package***

The move interface package was written to simplify the user's interface to MoveIt!. Like the NetFT utilities package, there is a version that allows access via ROS messaging and also a version that can be instantiated directly. GCCF does not pass messages through ROS messaging structures to the move interface package. Instead, GCCF instantiates the class and accesses it through several access methods of the class. With the Move Interface package, robot motion requires only a couple parameters (desired position and speed fraction) to define a desired motion. Move Interface also keeps track of a state of the robot so that users cannot command the robot when inappropriate, e.g. while a move is executing already or when the robot is in a FT violation state. Like previously mentioned, it also monitors the NetFT utilities package cancel message to know when the user defined maximum force or torque has been exceeded and requires a reset before further commanding to so the robot does not move when it is not safe. The Move Interface package also allows for very simple commanding of Cartesian moves, which is something that MoveIt! does not make easy. Another important feature is that the interface constantly publishes a status message that allows the user or code to gain insight as to why commands are not being executed, if a move has finished, and if an error occurred during the move.

#### **4.4 PUTTING IT ALL TOGETHER**

Figure 4.5 shows GCCF running in a demonstration. The arrows in the diagram represent information being passed from one package to another and, except for the double sided arrows showing the communication from the drivers to the hardware, are all managed by previously discussed ROS messaging structures. Note that some messages to and from standalone ROS packages are not shown, e.g. auxiliary packages sending messages to TF to update coordinate reference frames, and that some packages shown are simplified to just be one package even though in reality there is an interface from one package to another.

For example, the diagram shows the NetFT driver and NetFT utilities packages in one block and shows MoveIt! directly communicating with the Contact Control class even though the Move Interface package actually handles the interface between the two. The diagram is not meant to spell out each individual piece, but to show the task oriented groups of packages that the Contact Control class communicates with.

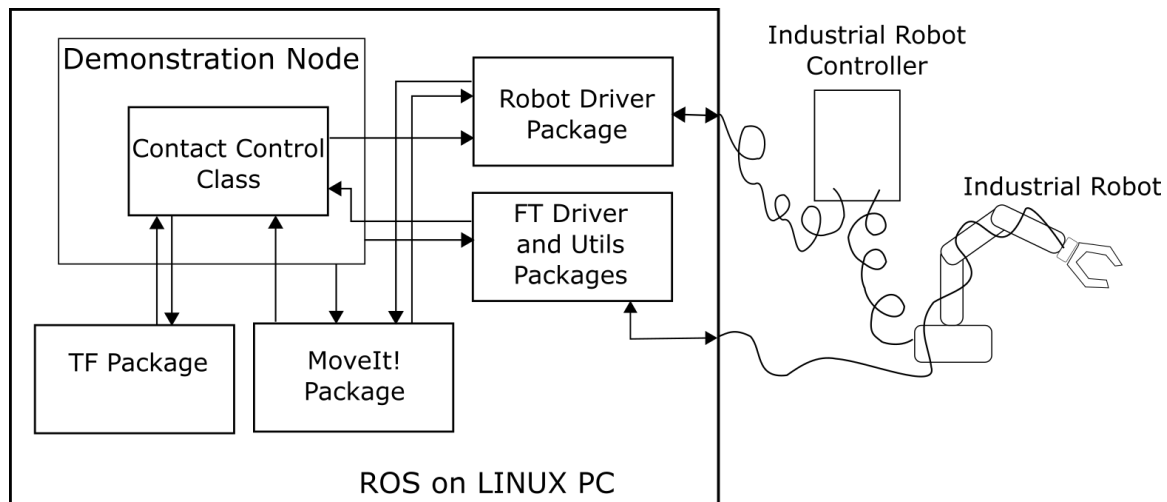


Figure 4.5 Implementation and Interfaces of the generalized contact control framework

As is shown by the large rectangle enclosing all of the boxes that represent groups of ROS nodes or packages, all of the code written for and utilized by GCCF was written in or embedded into ROS. Note that any hardware that is meant to communicate with ROS nodes needs a hardware specific ROS driver that communicates with the hardware and packages communications to and from the hardware into ROS message structures. For a selection of industrial hardware, these drivers have already been written and can be found on the ROS-Industrial website [52].

The contact control class in the diagram is the package containing all of the C++ code that makes up GCCF. GCCF is instantiated in the ROS node that runs the demonstration or robotic procedure. The demonstration node uses the previously discussed

access methods to tell the contact control class when to execute moves. The demonstration node also manages the motion of the robot when it is moving from task space to task space, loads collision objects into MoveIt!'s planning scene, interfaces with the end user via terminal or graphical user interface when needed, and requests that the force/torque sensor apply bias or threshold.

The contact control class itself communicates with the TF package by sending data to be transformed to different coordinate frames and receiving the transformed data. It also receives data from MoveIt! about the current pose of the EEF. It receives current force/torque readings from a force/torque driver or utility package and it also sends jogging commands directly to a robot driver which then relays them to the industrial robot controller.

#### **4.5 FINAL NOTES ON IMPLEMENTATION**

GCCF is implemented within the ROS architecture and written in C++. As stated previously, ROS integration allows the framework to remain hardware agnostic. Hardware agnosticism allows GCCF to be implemented on many different hardware platforms, but it does not necessarily make it easier to use. When a user applies GCCF to a task, they must first find or code the appropriate drivers that transform hardware communications into ROS friendly message structures. For example, GCCF sends jogging commands as a simple 0-1 fractional velocity where 1 is max speed and 0 is no speed. If necessary, an intermediate node may receive velocity commands to convert them into a more structured value, e.g. by adding units or converting to a trajectory stream, before passing them to the robot driver. Then the robot driver must encode and send this information to the industrial robot controller. Also, GCCF communicates with MoveIt! which is meant to be configurable for

any serial manipulator, but the user must write a universal robot description file, or URDF, which outlines the kinematic properties of the robot.

Also, note that GCCF's implementation is not real-time. Real-time control might be implemented when the functionality becomes available in ROS 2.0 [53]. For now, control is executed at 500hz or as fast as hardware allows. It is beyond the scope of this thesis to determine what rate is fast enough for smooth control, but a suggested rule of thumb is to select computing resources such that they can command the selected robot at its maximum available control rate. In other words, do not let the Linux machine that runs GCCF be the bottleneck for the overall system.

## Chapter 5

### Demonstrations of the Generalized Contact Control Framework

To demonstrate the validity and usefulness of GCCF, two demonstrations were designed. The first demonstration is a graphical user interface, or GUI, that is wrapped around the C++ code that runs GCCF. This interface visualizes the behavior of the framework and proves that the dimensions of a reference frame can be independently controlled. The second demonstration is an application demonstration involving multiple contact control moves. The application demonstration shows that GCCF can be used for actual contact task behaviors.

#### 5.1 HARDWARE INTEGRATION FOR DEMONSTRATION

To accomplish the two demonstrations of GCCF, we first needed to pick hardware and integrate it into a ROS driven system. As stated previously, the required components to run the framework are a Linux computer that is able to run ROS, a wrist mounted FT sensor, and an industrial robot and controller. To run these demonstrations, we used ROS-Indigo running in Ubuntu 14.04 on a Lenovo Y-50-70 laptop, but we could have used any Linux computer that runs ROS-Indigo. The only concern when choosing a computer, as stated previously, is that if the computer is introducing lag to the system, then there may be undesirable consequences when trying to control the contact with the environment, e.g. oscillatory behavior or movement well past desired force/torque cutoffs. To test the Linux machine, we ran GCCF and clocked the time between force data receipt and velocity commanding. With the laptop selected for our demonstrations, this process executes at around 500 Hz.

The demonstrations use a 6-axis ATI Gamma sensor [54], seen in Figure 5.1, to sense forces and torques at the wrist. This sensor is lightweight, high strength, and has

minimal noise. The sensor is also the proper shape to be wrist-mounted. The sensor sends data via the ATI Net F/T server to the Linux machine running ROS via Ethernet. The Net F/T server is capable of outputting data at 7000 Hz [55] which far exceeds typical industrial robot control rates. The server transmits 3 dimensions of forces and 3 dimensions of torques, so complete spatial control is possible.



Figure 5.1 ATI Gamma FT transducer [54]

The demonstrations use a Motoman SIA5 industrial robot shown in Figure 5.2. The SIA5 is a 7 DOF, 5kg payload industrial manipulator with  $\pm 0.06$  mm repeatability [56]. The fourth joint of the robot is offset 45 degrees to extend the workspace to be closer to the base of the robot. This is useful for contact task operations that involve manipulating items on a table close to the robot.





Figure 5.2 Motoman SIA5 7DOF Robot

The industrial robot controller is an Agile Planet AX controller. This controller is commanded via CeWin which is a real-time virtual machine that is run on a Windows computer [57]. This creates an interesting interface since, as stated previously, GCCF runs on a Linux machine due to its incorporation into ROS. To access robot control from Linux, a previous NRG student wrote a custom ROS driver that communicates with the windows machine to send commands to the controller. This driver takes ROS trajectory messages, sends them over a TCP connection to the CeWin system, and then relays them to the AX controller. Since GCCF outputs velocity commands, it was necessary for these demonstrations to add functionality to the driver to also be able to relay velocity commands to the controller. This is an example of the work a user of GCCF must complete to use the hardware agnostic interfaces with equipment that does not have pre-built ROS drivers, or

does not accept commands in the form that is outputted from GCCF. The completed hardware interface can be seen in Figure 5.3.

Note that the demonstrations command the robot through a real-time component even though GCCF does not run in real-time. This does not make the system real-time however, since GCCF does not schedule commands but produces them as fast as possible up to 500 Hz. But an important aspect of velocity commanding through the CeWin interface is that commands have a duration parameter. This parameter tells the robot how long to keep executing a velocity command if a new command is not received. For our demonstrations, the robot driver sets this parameter to 2 milliseconds so that the robot will not keep executing a velocity once GCCF is done commanding a move. It is important for a user of GCCF to implement such a feature when setting up a demonstration. This requires the user to characterize the nominal control rate of their selected system. The user must implement a duration parameter that is about the same, or possibly a little larger, than the time between consecutive commands. This ensures that the robot will not continue to move after commanding is halted and also will not stop and start between each command.

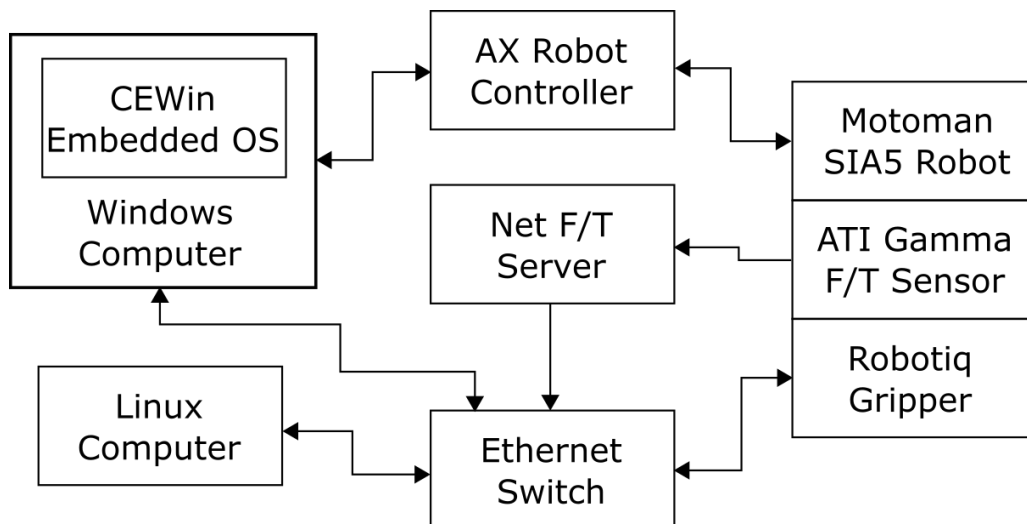


Figure 5.3 Demonstration Hardware Setup and Integration

Figure 5.3 shows that we also attached a Robotiq gripper to the robot for manipulation. This is not listed as a necessary component for the framework to work and it does not interface directly with GCCF, but any demonstration manipulating the environment requires some manipulation tool attached to the robot. The Robotiq will be controlled by the demonstration node, not by GCCF itself. The gripper we used for the demonstrations is a Robotiq S-model 3 finger gripper [58]. The Robotiq gripper, seen in Figure 5.4, attaches to the force/torque sensor at the end of the SIA5. The gripper already has ROS packages to interface with the gripper controller and is commanded from any ROS node running alongside the Robotiq driver packages.



Figure 5.4 Robotiq 3-Finger Gripper [58]

## 5.2 DYNAMICALLY RECONFIGURABLE DEMONSTRATION

The first demonstration shows the ability of GCCF to control each axis independently and the ease at which a user can change the control modes. The demonstration code is a GUI that wraps around the public access methods of GCCF. The GUI, seen in Figure 5.5, allows a user to set the control law and stiffness for each axis. The control laws and stiffness values can be set before a move, or updated in the middle of the

move. The demonstration should not control the robot during an actual contact control task. It is supposed to be illustrative of the ease of using the framework and the variety of control a user can achieve by simply adjusting the available parameters. It also allows a user to physically interface with the laws by applying force to the EEF. This gives the user an intuitive understanding of how the laws behave.

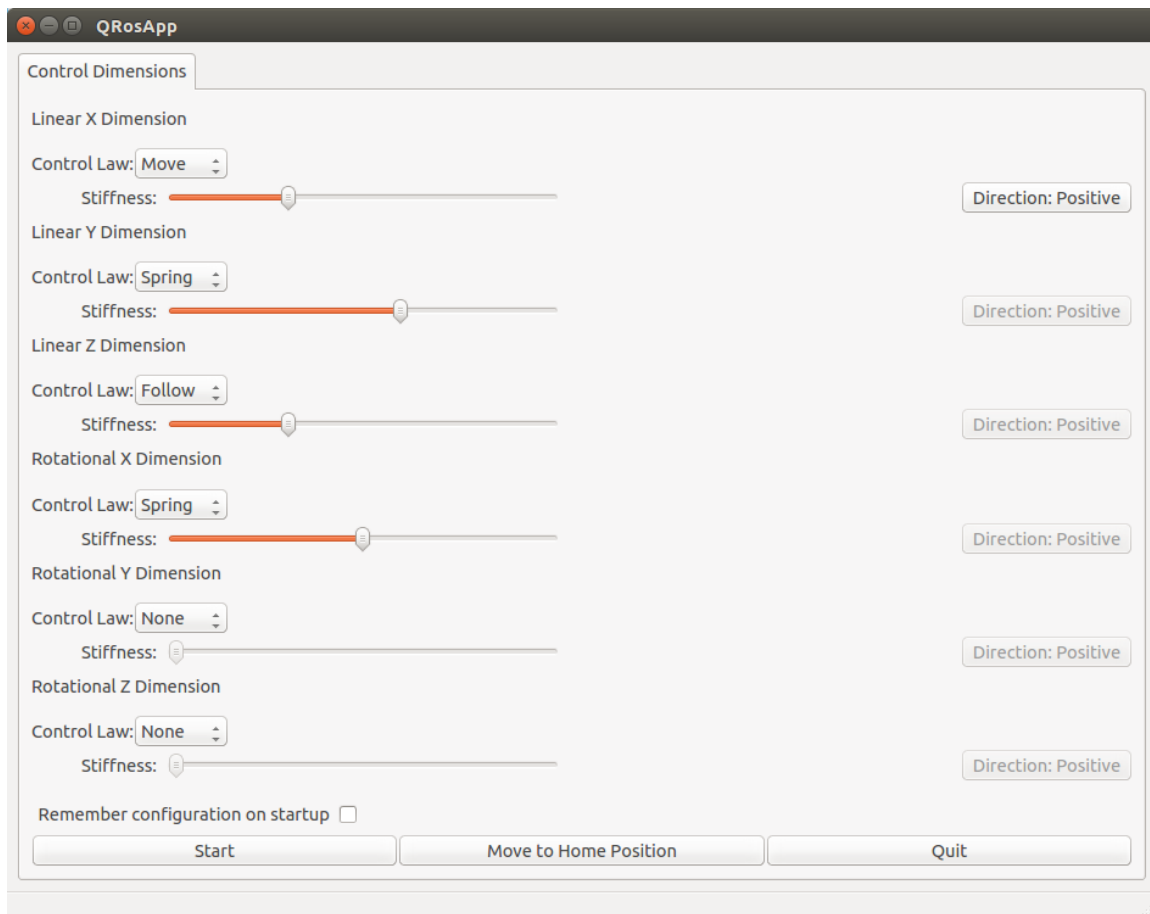


Figure 5.5 Dynamically Reconfigurable GUI. Each Axis has a drop down menu that allows the user to select desired control law (None, Move, Spring, or Follow) well as modify the stiffness parameter using a simple slider.

To use the dynamically reconfigurable GUI, a user must choose at least one dimension to be controlled. The user can control 6 dimensions in all:  $x$ ,  $y$ , and  $z$  linear

control and  $x$ ,  $y$ , and  $z$  rotational control. Once the user selects a control law for a dimension, they can adjust the stiffness bar to the desired stiffness. The stiffness does not have units on the GUI and is simply a low to high value. After picking a control law and stiffness for each axis to be controlled, the user presses the move button and a move starts. At any point after this, the user may end the move by pressing the stop button, or update the move by adjusting the laws and stiffness values and pressing the update button. While the move is executing, users can apply forces to the EEF to see the resulting behavior of the robot.

### **5.3 APPLICATION DEMONSTRATION**

To demonstrate an application of GCCF, a multistep task to a place peg in hole demonstration was conceived. The application demonstration requires multiple and varied contact tasks be completed in order to achieve the overall goal of putting a peg in a hole. In the task, the robot picks up a peg from the table and puts it in a peg hole of a peg board. The demonstration configuration can be seen in Figure 5.6.

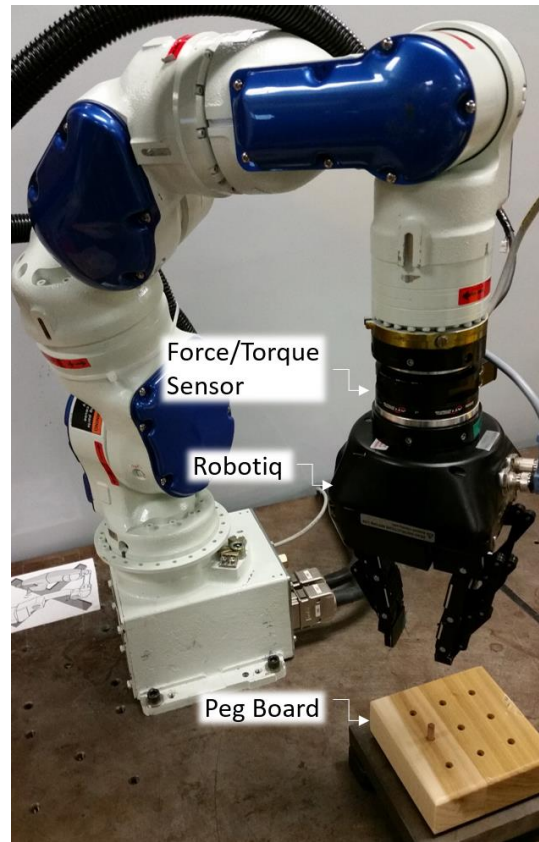


Figure 5.6 Peg in hole demonstration configuration

The demonstration involves three main tasks which are visualized in Figure 5.7. Some of the tasks require executing multiple moves using GCCF. The demonstration code does not know the precise position of the table, peg, and peg board and thus the uncertainty assumed for the task is well within the current capabilities of modern vision systems to estimate their pose. The demonstration executes all contact moves in the fashion described in the design and implementation chapters, i.e. with compliant moves and end conditions so that positions do not need to be precisely known.

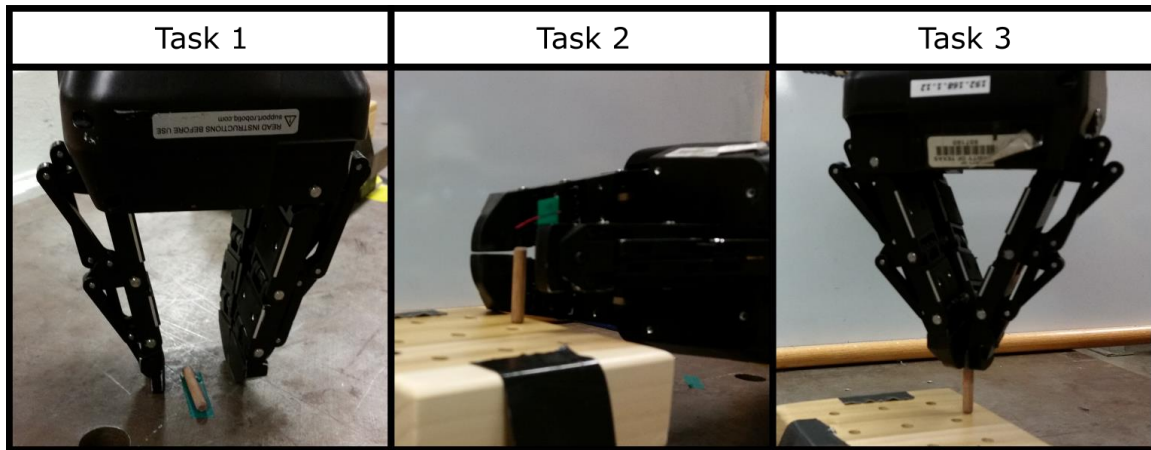


Figure 5.7 Tasks for peg in hole application demonstration: 1) grasping the peg, 2) balancing and re-grasping the peg, 3) tracking to the hole and inserting the peg

### 5.3.2 Demonstration Tasks

The first task is to pick up the peg from the table. This is a tricky task due to the geometry of the peg and the gripper. The peg has a diameter of approximately 6.3 mm, as seen in Figure 5.6 and Figure 5.7. As stated previously, the distance from the palm of the gripper to the fingertips does not stay constant while the gripper closes. This can be minimized by keeping the fingers open only just enough to grasp the peg, but this reduces the amount of uncertainty allowed when estimating the initial location of the peg on the table. This means that even after moving down to touch the table, the gripper fingers cannot simply close around the peg to pick up the peg from the table. In doing so, the gripper would apply large contact to the table if the distance between the palm and fingertips was increasing. In this case the fingers would not make it all the way to the closed position. If the distance between the palm and the fingertips was decreasing, the fingers would lose contact with the table and could either miss the peg completely or grasp it improperly. To solve this problem, GCCF allows the EEF of the robot to move vertically in relation to the

forces sensed to maintain contact between the fingers and the table while the fingers close without applying excessive force.

After lifting the peg off the table in task 1, the peg still cannot move directly into the hole. This, again, is a consequence of the peg and gripper geometries. The gripper fingers are wide in relation to the length of the peg and not much of the peg sticks out from between the fingers. To get the peg in the position to be placed in the hole, task 2 is a second manipulation of the peg in the gripper fingers. In task two the robot balances the peg upright on the peg board and releases it so that the it can then grasp it from the top.

Lastly, after the gripper correctly grasps the peg, task 3 is to put the peg in the hole. The robot follows the surface of the peg board while looking for a hole to place the peg in. Then, the robot senses the contact force from the walls of the hole or the lack of contact in the downward direction. Without extra commanding from the user, the robot moves downward into the hole instead of laterally across the peg board until the peg is completely in the hole.

### **5.3.3 Execution of the Demonstration Tasks**

To discuss how the demonstration uses GCCF to accomplish the tasks of the peg in hole procedure, this section breaks each task down into the separate moves that make up the task as a whole. Note from Figure 5.9 that between each main task, the demonstration executes moves from task space to task space. These moves will not be discussed thoroughly since they do not use GCCF. The moves between task spaces use MoveIt! to command the robot to move to a space believed to be above the next task space. During these moves, the demonstration loads conservatively sized collision objects, shown in Figure 5.8, into MoveIt!’s planning scene so that the generated trajectory will not take the robot near the environmental objects.



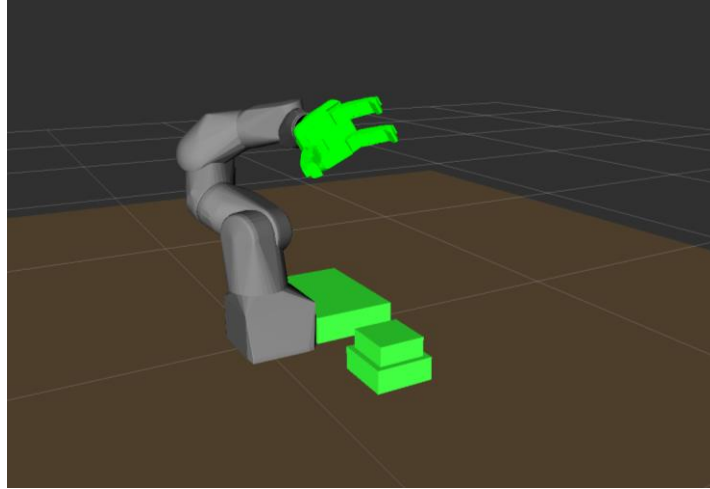


Figure 5.8 MoveIt! planning scene including collision objects

After the move to the task space, the demonstration also executes a move with the generalized contact control framework to get closer to the task before interacting with the environment. A summary of the moves that will be explained in the next few sections is shown in Figure 5.9.

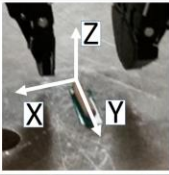


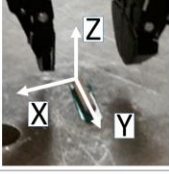


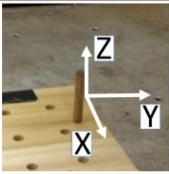
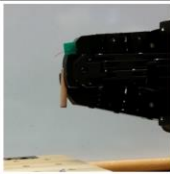
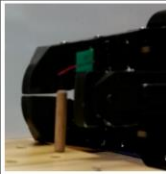
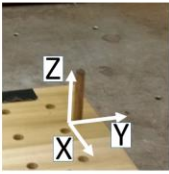
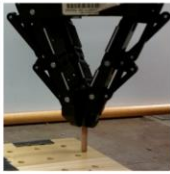
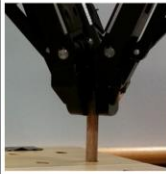
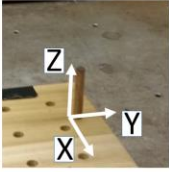
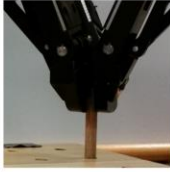
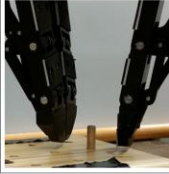
Move	Reference Frame	Control Laws	Start	Finish
1		Z: Movement		
2		Z: Spring		
3	Non-contact move up			
4	ROS MoveIt move to next task space			
5		Z: Movement		
6	Non-contact move up			
7	ROS MoveIt move to next task space			
8	Non-contact move down to grasp			
9		X: Movement or spring Y: Movement or spring Z: Spring		
10		X: Movement or spring Y: Movement or spring Z: Movement		

Figure 5.9 Peg in hole demonstration tasks

### 5.3.3.2 Execution of Task 1

The execution of task 1 breaks down into three moves which are shown as moves 1-3 in Figure 5.9. The EEF first moves down until contacting the table with the gripper open around the peg. Then the gripper closes while the EEF adjusts vertical position to

maintain contact, and finally the EEF safely moves up away from the table while holding the peg. The reference frame that describes the moves is defined in Figure 5.10.

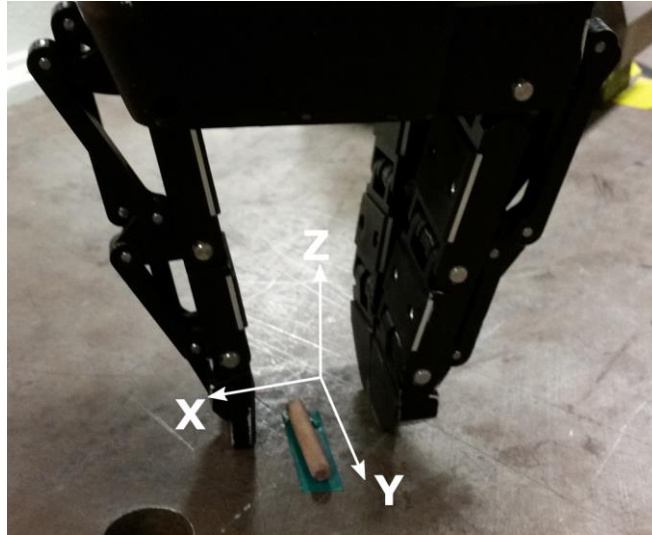


Figure 5.10 User defined coordinate reference frame for task 1 moves

For the first move, the downward move to contact the table, GCCF only controls the  $z$  dimension pictured in Figure 5.10. GCCF sets the dimension as a compliant move in the negative  $z$  direction. The end condition for this move is a maximum force. Since the robot does not know the distance to the table, the only way to know when to stop is by monitoring the force in the  $z$  dimension. Note that if the angle of incline along the  $x$  axis was unknown, the user could also set a follower or spring law around the rotation of the  $y$  axis to ensure that all fingers contact the table. Since the demonstration robot is mounted on the table, we are certain of the slope of the table and do not need to apply these extra laws.

The second move is the scraping grasp that captures the peg in the gripper. Again, GCCF only controls the  $z$  dimension for the second move. The demonstration sets the  $z$ -dimension to follow the spring law with an unstretched offset of 4 millimeters into the

table. This law allows the gripper to move upward if the fingers are pushing too hard on the table but it will always move downward towards the spring's equilibrium position, if possible. This robot executes the move asynchronously after the demonstration code commands the gripper to close and then finishes once the gripper signals that it is closed. The robot does not control rotational dimensions in the second move because it is important that the gripper picks up the peg orthogonally to the palm normal. This requirement ensures that when the robot stands the peg upright in the second move it is in the right orientation. After the gripper has finished closing, the demonstration executes a final compliant move in the positive  $z$  dimension to retreat from the task space and allow for a planned trajectory to the next task space.

#### ***5.3.3.3 Execution of Task 2***

Task two, shown as moves 5 and 6 in Figure 5.9, is to balance the peg on the side of the pegboard. The robot executes this task with two compliant moves oriented along the  $z$  dimension in Figure 5.11. For the first move, GCCF controls the negative  $z$  direction with a compliant move law. There is also a spring law on the rotational  $y$  dimension and a spring law with an adjustment for the lever arm to the sensor on the rotational  $x$  dimension. The spring laws compensate for very small inaccuracy of the peg rotation. By tweaking the lever arm on the  $y$  dimension, it could be possible to balance a slightly rotated peg, but due to the nature of the way that the peg is picked up in task 1, this turns out to be unnecessary. After the first move, the gripper opens to release the peg. At this point, a vision integrated system could determine if the peg remained standing after the release so that it can be re-picked if it fell over, but for this demonstration we have not yet incorporated this functionality. After the peg is released, the robot executes the second move. This move is a 1-dimensional compliant move in the  $z$  dimension to move away from the task space and

allow for motion to the final task space. Note that an important feature of the move from the second to third task space is that the demonstration code adds the peg to the collision scene on the peg box so that the planned move will not knock over the peg.

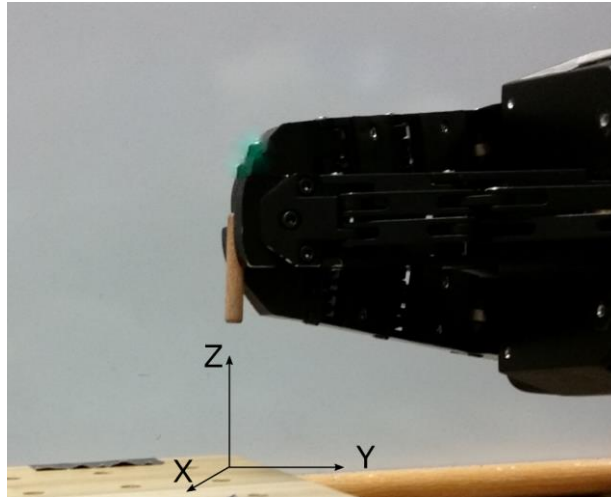


Figure 5.11 User defined coordinate reference frame for task 2 moves

#### ***5.3.3.4 Execution of Task 3***

The third task, shown as moves 8-10 in Figure 5.9, is to put the peg into one of the holes on the pegboard. The first move of the task starts with the gripper open above the peg and moves downward to position the fingers around the peg. This move is a one dimensional, non-contact move to closer proximity to the task space similar to those discussed before. Then, the gripper closes to grasp the peg which puts the EEF in direct contact with the environment in the state shown in Figure 5.12.

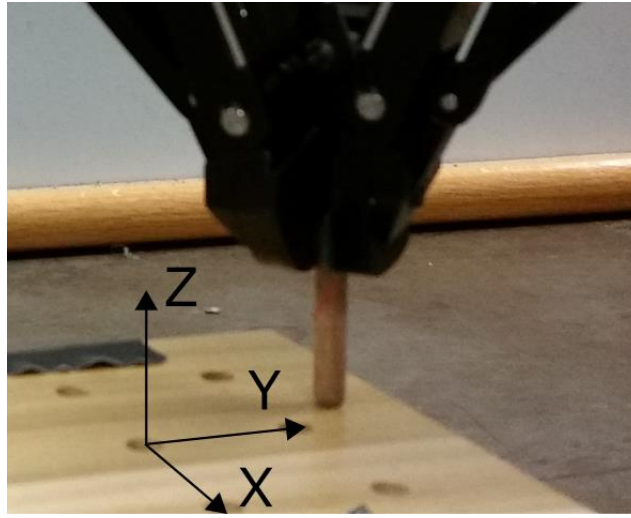


Figure 5.12 User defined coordinate reference frame for task 3 moves

After the first move, the robot drags the peg along the peg board to find a hole. Since there is no vision incorporated into this demonstration, the robot has no knowledge of where the holes are on the peg board. For this reason, the demonstration relies on a small amount of teleoperation to determine the direction of motion. The demonstration sets up the move with 3 controlled dimensions. The demonstration sets the  $z$  dimension as a compliant move in the negative  $z$  direction (into the peg board). This control maintains contact with the peg board as long as there is a contact force resisting motion, but once the peg is above a hole and the contact force disappears, the EEF will start moving downward into the hole without the demonstration changing control laws. The demonstration sets control of the linear  $x$  and  $y$  as one of two laws. One of these dimensions is set as a compliant move. Whichever dimension is not set as a compliant move is set as a spring. The user tele-operates which direction is set to compliant move in real-time while the demo is running by pressing keys on the Linux machine's keyboard. For example, in the case shown in Figure 5.12, the user might first choose motion in the negative  $y$  direction. By

hitting the appropriate key, the user sets the  $y$  dimension to a compliant move in the negative  $y$  direction, and the  $x$  dimension to a spring with no offset. Once the peg moves in to alignment with the hole in the  $y$  dimension, the user can then hit the appropriate key to start motion in the  $x$  dimension. In response, the demonstration code sets the  $x$  dimension to move positively as a compliant move law and the  $y$  dimension to be controlled as a spring. All the while, GCCF is still executing the  $z$  dimension compliant move control. Once the peg is slightly over any of the holes, the normal forces caused by the peg walls direct movement in the  $y$  dimension (compliant move law) and the  $x$  dimension (spring law) towards the center of the hole until there is enough clearance that the  $z$  dimension will sense a smaller force and motion will start downward. The end condition on this move is a small displacement in the negative  $z$  direction so that the control laws can be adjusted for a smoother move once the peg has started into the hole. After this move is done, the demonstration sets up the final move. For the final move, the  $z$  dimension remains as a compliant move downward, and the  $x$  and  $y$  dimensions behave as force follow laws so that they will resist movement that increases contact with the peg hole walls. The two important end conditions for this move are displacement downwards and force in the  $z$  dimension. If contact causes a large force in the  $z$  dimension, the peg has hit the bottom of the peg hole, or the fingers have hit the pegboard surface and the move is done. If the robot has moved far enough into the hole so the peg is secure in the hole, the move is done. After this move, the demonstration code opens the gripper and uses a compliant move to retreat from the task space and the demonstration is complete.

#### **5.4 SUMMARY OF DEMONSTRATIONS**

The dynamically reconfigurable GUI demonstration successfully showed the flexibility and modularity of GCCF by allowing a user to pick separate control laws for

each linear and rotational dimension. The user is able to gain a physical interpretation of the function of the control laws and the behavior associated with setting control on multiple dimensions. The demonstration also shows off the ability of GCCF to be used in co-robotics applications by allowing the user to interact directly with the EEF.

The peg in hole application demonstration proved that GCCF can be used to complete actual contact tasks, including grasping, manipulation, and assembly. The demonstration also proved that these contact tasks could be done safely within the bounds of allowable stress on the robot. Due to the orthogonal nature of the demonstration components, the demonstration was mostly limited to control on linear axes and did not show off GCCF's ability to control rotational axes in the same way as linear axes.

Through the grasping of the peg from the table surface, we learned that GCCF behaves better at slower speeds. It was noticed that when the Robotiq fingers were set at a higher speed, it was more difficult to tune the controller to maintain contact with the table. Note that precise parameter values were not given in the discussion of the demonstration, but they may be found in Appendix B. A future version of the demonstration could incorporate vision monitoring and introduce fault procedures to show off the ability of GCCF to perform in uncertain conditions.



## **Chapter 6**

### **Conclusion and Future Work**

#### **6.1 RESEARCH SUMMARY**

Although robotic capabilities in research labs and particular industry applications have improved greatly since the first years of reprogrammable robotic control, many industries, and especially the nuclear industry, still limit the use of robotics to simple pick and place, non-contact tasks. And in cases when contact-tasks are considered, integration costs are typically much higher. The task space of an industrial robot could be greatly increased in these fields by expanding control to include contact task procedures using a controller that is easily accessible and reconfigurable.

The generalized contact control framework, GCCF, presented in this thesis allows a programmer to complete a configurable contact control or co-robotics task with little knowledge of the field of compliant robotic control. GCCF has been implemented in a safe, accessible, and hardware agnostic manner using the Robot Operating System, ROS, and object oriented design. The design and implementation of the framework will expand the available task space of an industrial robot by allowing for uncertainty in models and mitigation of dangerous contact forces.

The ease of use and practicality of the framework were shown in two demonstrations conducted at the Nuclear and Applied Robotics Lab at the University of Texas at Austin. The first demonstration wraps the GCCF code in a graphical user interface that allows a user to change the control scheme of each dimension of a user defined reference frame. The user applies a force to the EEF of the robot and can gain a physical appreciation of how GCCF works. The second demonstration uses GCCF to perform multiple contact-task operations to pick up a peg, manipulate the peg to be handled in a

different orientation, and place the peg into a hole of a peg board. These demonstrations proved that the framework was feasible, and that it could be used to conduct multiple contact tasks within safe maximum force and torque limits.

## **6.2 RECOMMENDATIONS FOR FUTURE WORK**

The primary goal of this research was to make contact tasks easier to implement and this was achieved by developing a framework used in the demonstrations outlined above. As new contact tasks are envisioned and/or attempted with GCCF, foreseen and unforeseen challenges will arise. While accomplishing the work presented in this thesis, many avenues for future work were discovered.

### **6.2.1 Position Commanding**

Currently, a user commands GCCF one step at a time by setting up control laws for each dimension and executing a move until an end condition is met. Typically, the user sets most dimensions to be moved only if there is an external force sensed, and sets one or two dimensions to the compliant move law so that they will execute the desired motion. The compliant move law takes a desired velocity and attempts to execute that velocity until an end condition is met. The control law parameters of each dimension can be updated while the move is executing, as seen in the application demonstration when the operator uses teleoperation to update the move direction to find a peg hole.

To extend the capabilities of GCCF, a position controller, shown in Figure 6.1, could be implemented around the framework that constantly updates the desired velocity parameter to the compliant move law on one or multiple dimensions. Using this controller, a user could input a desired position and control parameters for each dimension. Using this desired position and the actual position from the robot, the position controller could update

the desired velocity of the compliant move laws to get the manipulator to the desired position.

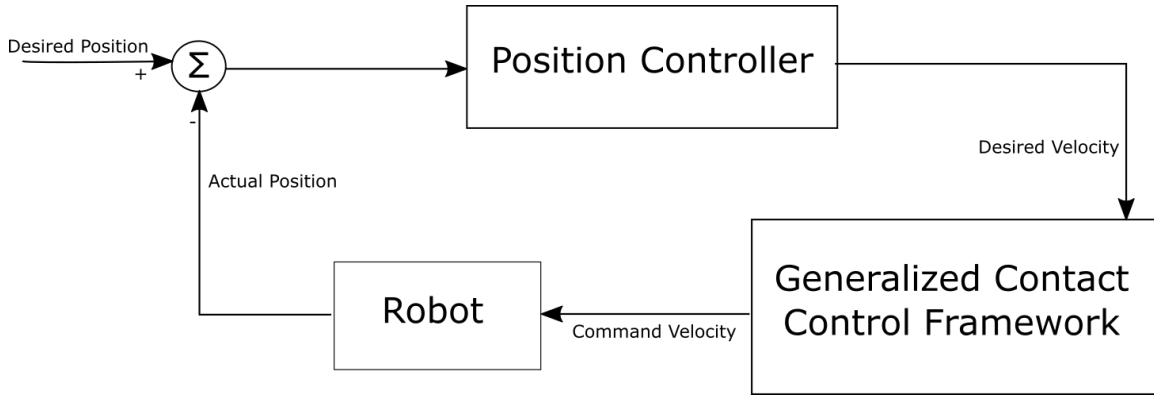


Figure 6.1 A position controller implementation

## 6.2.2 More Control Laws

The current version of GCCF is set up with three simple control laws that make it easy for an unexperienced operator to perform simple tasks. However, there is no reason that more control laws could not be implemented to cover less generic situations. The inexperienced user could still stick to the three useful laws presented in this paper, and any experienced user could apply more complicated laws. Some examples for possible additions to control capabilities follow.

### 6.2.2.1 *Explicit Force Control Law for Tasks that Require a Precise Applied Force*

The current state of GCCF safely mitigates contact forces while trying to follow the velocity trajectory specified by the user. Some force control procedures require more than safe operation. To apply a constant force or pressure on an object, an explicit force control law may be added to GCCF. This could expand the capabilities of GCCF to procedures such as polishing or painting an unknown surface without variations in brushstroke.

#### ***6.2.2.2 Adaptive Control Law***

As discussed in the literature review of this thesis, there have been many efforts in the literature to develop adaptive compliant control laws that can deal with changing environmental stiffness. Such a law could be implemented in GCCF as an extra option for controlling the EEF. This would enhance GCCF's capability to deal with uncertain environments.

#### ***6.2.2.3 Other Variations of Impedance Control Laws***

The impedance control laws used for the current version of GCCF are simplifications of a full impedance law. It might be useful to expand these laws into the full realization of the impedance control law described by Hogan and discussed in the literature review of this thesis. It is important to note that the law would need to be edited to be velocity based rather than position based since GCCF sends velocities to the robot controller. This new law would need more parameters, which might make it harder to implement. The literature review of this thesis showed that there are many versions of impedance control and there might be more that are suitable for GCCF.

#### **6.2.3 Modular Control Law Editor**

The idea of being able to add extra control laws leads into the next idea for future work which is to add the capability for a modular input of control laws. By making a control law editor that outputs a configuration file to be read in by GCCF at start-up, GCCF's capabilities could be extended without having to continuously edit the source code of the framework. The editor could have certain design variables available to the user crafting the control law such as displacement from start position, sensed force/torque, actual velocity, time passed, etc. The user could pick the names for variable parameters that need to be passed to the framework when implementing the law. Using these design variables and

variable parameters, the user can craft an equation that relates the variables to the outputted command velocity.

#### **6.2.4 Adopting the Control Framework to Other Challenging Applications**

Key to the success and improvement of GCCF is its usefulness to complete tasks at LANL, for manufacturing, or in any situation where an industrial robot may be called upon to complete contact tasks in the presence of uncertainty. Future work should include attempts to implement GCCF on practical applications. Some possible applications include:

- Opening doors and cabinets
- Dual arm robotics
- Surface finishing
- Station scheduling
- Material reduction
- Assembly/packaging

The first step to proving that GCCF can be used on a wide variety of contact tasks could be to reproduce previous NRG efforts to solve these tasks with robots. Figure 6.2 shows 3 applications that have previously been demonstrated by NRG. The top picture shows a material reduction demonstration accomplished by Peterson [12] that was detailed in the introduction of this thesis. GCCF could be used to mitigate forces applied to the EEF by the hole punch that is used to size reduce objects. The middle picture shows two robots working together to hold and move an egg. This demonstration was accomplished with the use of a fuzzy control law that adjusted the velocities of the two EEFs [59]. This demonstration could be attempted using GCCF separately on both arms. The first attempt should use the original control laws of GCCF by setting one robot to move compliantly on

multiple axes and one to follow on all axes. If this does not work, the original work could be replicated by adding a fuzzy logic law to GCCF.



Figure 6.2 Future applications of GCCF. Top: A robot size-reduces a bowl [12]. Middle: Two robots hold an egg [59]. Bottom: A robot opens a cabinet door [60].

Opening a cabinet door is another challenge that could possibly be solved with GCCF. A set of guarded moves could identify the exact location of the handle. One last guarded move could position the EEF to grasp the handle. A multi axis move could then open the door by compliantly moving in the direction away from the cabinet surface and

following in the orthogonal direction. The move would also implement a follower on the rotational hinge dimension.

### **6.2.5 Task Space Formalisms**

One question beyond the scope of this effort was to ascertain the breadth of tasks that could be completed using GCCF. To do this in a formal matter, first the *task* must be formally defined to truly encapsulate all real tasks and a comprehensive subset must be determined before full consideration of the applicability of the control law can be examined. It is possible that such formalisms already exist in the literature, but if not, the task formalisms must be developed.

### **6.2.6 Additional Hurdles**

The integration of contact control into an industrial robot framework was just one of the many hurdles described in the introduction of this thesis that are limiting the ability of industrial robots to be able to replace and augment humans in the workforce to improve safety and efficiency. Many other hurdles are still awaiting integration such as precise 6 DOF pose estimation from a vision sensor, reliable grasp validation, faster and more direct path planning, direct integration of the sensed environment with the collision scene, etc.

## **6.3 CONCLUDING REMARKS**

The proposed framework takes a large step towards tackling the hurdle of making contact control capabilities readily available to an industrial robot framework. With this framework, and possibly some of the additions described as recommended future work, industrial robots can begin to accomplish a larger variety of task, while remaining practical for short run task. With the improvement and integration of the other hurdles to robotic capabilities discussed previously, the task space of industrial robots will be dramatically increased.

## Appendix A

### Example C++ Code for Asynchronous Move

```
#include "ros/ros.h"
#include "contact_control.h"

// C++11 future must be included to use moveAsync
#include <future>

.....

// Instantiate contact control class
ContactControl cc;

// Initialize contact control class.
// All arguments are string configuration variables that should have been instantiated
// and set before call to initialize
cc.initialize(moveGroup,worldFrame,ftFrame,controlFrame,jogger,ftAddress);

.....

// This variable allows access to the return value in the future
std::future<Contact::EndCondition> ec;

// Set up control dimensions. At least one dimension must be configured to perform a move
cc.setSpring(Contact::DIM_Z, 20.0, 50.0, 0.004, 50.0);

// Return variable must be stored for C++11 async to work
ec = cc.moveAsync(70.0,15.0,0.9);

// Stop the move after 3 seconds
ros::Duration(3.0).sleep();
cc.stopMove();

// get() function must be called for C++11 async to work
// This function will block until the move is done, but since we called
// stopMove already, the move should be ended.
Contact::EndCondition endCon = ec.get();

// Print out the end condition. If nothing went wrong before the stop was called
// end condition will be enum EXTERNAL. Otherwise it will be the reason for the stop.

ROS_INFO_STREAM("End condition of async move: " << endCon);
```



## Appendix B

### Design Parameters for Peg in Hole Demonstration

Move Number	1	2	3	5	6	8	9			10		
Control Law*	M	S	M	M	M	M	M	M	S	M	F	F
Move Direction	Z	Z	Z	Z	Z	Z	Z	X**	Y**	Z	X	Y
Speed (fraction)	-0.04		0.3	-0.03	0.3	-0.2	-0.2	0.02		-0.05		
Stall force or torque (N or N-m)	0.5		10	4	15	4	4	2		5		
K		20							40			
B		50							30		30	30
Position offset (m)		0.004							0			
Displacement maximum (m)	0.1		0.07	0.1	0.07	0.08	0.023	0.1		0.02		
Force or torque maximum (N or N-m)	0.8	50	50	30	15	4	40	40	40	10	40	40

\*M stands for Compliant Move Law, S for Spring Law, F for Follow Law

\*\*These directions can be swapped via teleoperation

## References

- [1] M. W. Spong and M. Fujita, “Control in robotics,” *Impact Control Technol. Overv. Success Stories Res. Chall.*, 2011.
- [2] J. G. C. Devol, “Programmed article transfer,” US2988237 A, 13-Jun-1961.
- [3] “Boston Dynamics: Dedicated to the Science and Art of How Things Move.” [Online]. Available: [http://www.bostondynamics.com/robot\\_Atlas.html](http://www.bostondynamics.com/robot_Atlas.html). [Accessed: 04-Apr-2016].
- [4] “Amazon Prime Air.” [Online]. Available: <http://www.amazon.com/b?node=8037720011>. [Accessed: 04-Apr-2016].
- [5] “Drone Catcher Drone Fires Nets At Lesser Drones,” *Popular Science*. [Online]. Available: <http://www.popsci.com/drone-catcher-drone-fires-nets-at-lesser-drones>. [Accessed: 04-Apr-2016].
- [6] W. R. Algar, K. Susumu, J. B. Delehanty, and I. L. Medintz, “Semiconductor Quantum Dots in Bioanalysis: Crossing the Valley of Death,” *Anal. Chem.*, vol. 83, no. 23, pp. 8826–8837, Dec. 2011.
- [7] D. Butler, “Translational research: Crossing the valley of death,” *Nat. News*, vol. 453, no. 7197, pp. 840–842, Jun. 2008.
- [8] I. A. Sucan and S. Chitta, “MoveIt! Concepts.” [Online]. Available: <http://moveit.ros.org/documentation/concepts/>. [Accessed: 29-Feb-2016].
- [9] “ATI Industrial Automation: Automatic / Robotic Tool Changers.” [Online]. Available: [http://www.ati-ia.com/products/toolchanger/robot\\_tool\\_changer.aspx](http://www.ati-ia.com/products/toolchanger/robot_tool_changer.aspx). [Accessed: 06-Apr-2016].
- [10] “Home - TakkTile.” [Online]. Available: <http://www.takktile.com/>. [Accessed: 06-Apr-2016].
- [11] M. O. | J. S. Writer, “Updated: ‘Steps’ toward pit production made at Los Alamos.” [Online]. Available: <http://www.abqjournal.com/710234/news/steps-toward-pit-production-made-at-los-alamos.html>. [Accessed: 17-Apr-2016].
- [12] C. D. I. Peterson, “Industrial automation and control in hazardous nuclear environments,” University of Texas, Austin, Tex., 2015.
- [13] M. Quigley, B. Gerky, and W. D. Smart, *Programming Robots with ROS, a Practical Introduction to the Robot Operating System*, First. O’Reilly Media, Inc., 2015.
- [14] S. Y. Nof, *Handbook of Industrial Robotics*, vol. 1. John Wiley & Sons, 1999.
- [15] A. Armagnac, “New Factory Worker: Teachable Robot Can Remember 200 Commands,” *Popular Science*, vol. 181, no. 2, pp. 79–81, Aug-1962.
- [16] D. E. Whitney, “Force Feedback Control of Manipulator Fine Motions,” *J. Dyn. Syst. Meas. Control*, vol. 99, no. 2, pp. 91–97, Jun. 1977.
- [17] “ATI Industrial Automation: Compliance Devices.” [Online]. Available: [http://www.ati-ia.com/Products/compliance/compensator\\_main.aspx](http://www.ati-ia.com/Products/compliance/compensator_main.aspx). [Accessed: 30-Mar-2016].
- [18] J. De Schutter, D. Torfs, and H. Bruyninckx, “Robot force control with an actively damped flexible end effector,” *Robot. Auton. Syst.*, vol. 19, no. 2, pp. 205–214, Dec. 1996.

- [19] G. A. Pratt and M. M. Williamson, "Series elastic actuators," in *1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95. "Human Robot Interaction and Cooperative Robots"*, *Proceedings*, 1995, vol. 1, pp. 399–406 vol.1.
- [20] M. Quigley, A. Asbeck, and A. Ng, "A low-cost compliant 7-DOF robotic manipulator," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 6051–6058.
- [21] G. Tonietti, R. Schiavi, and A. Bicchi, "Design and Control of a Variable Stiffness Actuator for Safe and Fast Physical Human/Robot Interaction," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005*, 2005, pp. 526–531.
- [22] E. G. Evan Ackerman, "How Rethink Robotics Built Its New Baxter Robot Worker," 18-Sep-2012. [Online]. Available: <http://spectrum.ieee.org/robotics/industrial-robots/rethink-robotics-baxter-robot-factory-worker>. [Accessed: 30-Mar-2016].
- [23] M. T. Mason, "Compliance and Force Control for Computer Controlled Manipulators," *IEEE Trans. Syst. Man Cybern.*, vol. 11, no. 6, pp. 418–432, Jun. 1981.
- [24] M. H. Raibert and J. J. Craig, "Hybrid Position/Force Control of Manipulators," *J. Dyn. Syst. Meas. Control*, vol. 103, no. 2, pp. 126–133, Jun. 1981.
- [25] H. Lipkin and J. Duffy, "Hybrid Twist and Wrench Control for a Robotic Manipulator," *J. Mech. Transm. Autom. Des.*, vol. 110, no. 2, pp. 138–144, Jun. 1988.
- [26] J. M. Selig and P. R. McAree, "A simple approach to invariant hybrid control," in *, 1996 IEEE International Conference on Robotics and Automation, 1996. Proceedings*, 1996, vol. 3, pp. 2238–2245 vol.3.
- [27] K. P. Jankowski and H. A. ElMaraghy, "Constraint Formulation for Invariant Hybrid Position/Force Control of Robots," *J. Dyn. Syst. Meas. Control*, vol. 118, no. 2, pp. 290–299, Jun. 1996.
- [28] J. K. Salisbury, "Active stiffness control of a manipulator in cartesian coordinates," in *1980 19th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes*, 1980, pp. 95–100.
- [29] H. Seraji, "Adaptive admittance control: an approach to explicit force control in compliant motion," in *, 1994 IEEE International Conference on Robotics and Automation, 1994. Proceedings*, 1994, pp. 2705–2712 vol.4.
- [30] N. Hogan, "Impedance Control: An Approach to Manipulation: Part II—Implementation," *J. Dyn. Syst. Meas. Control*, vol. 107, no. 1, pp. 8–16, Mar. 1985.
- [31] R. Anderson and M. W. Spong, "Hybrid impedance control of robotic manipulators," *IEEE J. Robot. Autom.*, vol. 4, no. 5, pp. 549–556, Oct. 1988.
- [32] G. J. Liu and A. A. Goldenberg, "Robust hybrid impedance control of robot manipulators," in *, 1991 IEEE International Conference on Robotics and Automation, 1991. Proceedings*, 1991, pp. 287–292 vol.1.
- [33] S. Jung, T. C. Hsia, and R. G. Bonitz, "Force tracking impedance control of robot manipulators under unknown environment," *IEEE Trans. Control Syst. Technol.*, vol. 12, no. 3, pp. 474–483, May 2004.

- [34] V. Mallapragada, D. Erol, and N. Sarkar, "A New Method of Force Control for Unknown Environments," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 4509–4514.
- [35] Y. Karayiannidis, G. Rovithakis, and Z. Doulgeri, "Force/position tracking for a robotic manipulator in compliant contact with a surface using neuro-adaptive control," *Automatica*, vol. 43, no. 7, pp. 1281–1288, Jul. 2007.
- [36] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, "Learning variable impedance control," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 820–833, Jun. 2011.
- [37] L. Kwang-Kyu, "Force Tracking Impedance Control with Variable Target Stiffness," 2008, pp. 6751–6756.
- [38] M. Jin, S. H. Kang, and P. H. Chang, "Robust Compliant Motion Control of Robot With Nonlinear Friction Using Time-Delay Estimation," *IEEE Trans. Ind. Electron.*, vol. 55, no. 1, pp. 258–269, Jan. 2008.
- [39] S. H. Kang, M. Jin, and P. H. Chang, "A Solution to the Accuracy/Robustness Dilemma in Impedance Control," *IEEEASME Trans. Mechatron.*, vol. 14, no. 3, pp. 282–294, Jun. 2009.
- [40] R. V. Patel, H. A. Talebi, J. Jayender, and F. Shadpey, "A Robust Position and Force Control Strategy for 7-DOF Redundant Manipulators," *IEEEASME Trans. Mechatron.*, vol. 14, no. 5, pp. 575–589, Oct. 2009.
- [41] R. Kikuuwe, "A Sliding-Mode-Like Position Controller for Admittance Control With Bounded Actuator Force," *IEEEASME Trans. Mechatron.*, vol. 19, no. 5, pp. 1489–1500, Oct. 2014.
- [42] "NRC: Glossary -- ALARA." [Online]. Available: <http://www.nrc.gov/reading-rm/basic-ref/glossary/alara.html>. [Accessed: 30-Mar-2016].
- [43] C. J. Turner, T. A. Harden, and J. A. Lloyd, "Robotics in Nuclear Materials Processing at LANL: Capabilities and Needs," pp. 701–710, Jan. 2009.
- [44] Robert Bogue, "Robots in the nuclear industry: a review of technologies and applications," *Ind. Robot Int. J.*, vol. 38, no. 2, pp. 113–118, Mar. 2011.
- [45] B.L. Luk, K.P. Liu, A.A. Collie, D.S. Cooke, and S. Chen, "Tele-operated climbing and mobile service robots for remote inspection and maintenance in nuclear industry," *Ind. Robot Int. J.*, vol. 33, no. 3, pp. 194–204, May 2006.
- [46] S. Tibrea, T. Nance, and E. Kriikku, "Robotics in Hazardous Environments - Real Deployments by the Savannah River National Laboratory," *J. S. C. Acad. Sci.*, vol. 9, no. 1, pp. 5–8, Mar. 2011.
- [47] T. A. Harden, K. A. Schroeder, and M. Pryor, "On the Use of Joint Torque Sensors for Collision Detection in a Confined Environment," Los Alamos National Laboratory (LANL), LA-UR-11-02804; LA-UR-11-2804, May 2011.
- [48] J. F. J. R. L. Kress, "The evolution of teleoperated manipulators at ORNL," *Proc 79th Top. Meet. Robot. Remote Syst.*, pp. 623–631, 1997.
- [49] A. Zelenak, C. Peterson, J. Thompson, and M. Pryor, "The Advantages of Velocity Control for Reactive Robot Motion," p. V003T43A003, Oct. 2015.
- [50] "Documentation - ROS Wiki." [Online]. Available: <http://wiki.ros.org/>. [Accessed: 25-Apr-2016].

- [51] “C++11 async tutorial | Solarian Programmer.” [Online]. Available: <https://solarianprogrammer.com/2012/10/17/cpp-11-async-tutorial/>. [Accessed: 18-Apr-2016].
- [52] “ROS-Industrial,” *GitHub*. [Online]. Available: <https://github.com/ros-industrial>. [Accessed: 15-Apr-2016].
- [53] “Design.” [Online]. Available: <http://design.ros2.org/>. [Accessed: 18-Apr-2016].
- [54] “ATI Industrial Automation: F/T Sensor Gamma.” [Online]. Available: [http://www.ati-ia.com/products/ft/ft\\_models.aspx?id=Gamma](http://www.ati-ia.com/products/ft/ft_models.aspx?id=Gamma). [Accessed: 15-Apr-2016].
- [55] “ATI Industrial Automation: Net F/T.” [Online]. Available: [http://www.ati-ia.com/products/ft/ft\\_NetFT.aspx](http://www.ati-ia.com/products/ft/ft_NetFT.aspx). [Accessed: 23-Mar-2016].
- [56] “SIA5D.” [Online]. Available: <http://www.motoman.com/datasheets/SIA5D.pdf>. [Accessed: 23-Mar-2016].
- [57] “CeWin [Windows® real-time platform].” [Online]. Available: <http://www.acontis.com/eng/products/windows-real-time-hypervisor/cewin/index.php>. [Accessed: 21-Apr-2016].
- [58] “3-Finger Adaptive Robot Gripper,” *Robotiq*. [Online]. Available: <http://robotiq.com/products/industrial-robot-hand/>. [Accessed: 23-Mar-2016].
- [59] A. J. Zelenak, “A compliant control law for industrial, dual-arm manipulators,” Jun. 2013.
- [60] K. A. Schroeder, “On the use of generalized force data for kinematically controlled manipulators,” thesis, 2011.