REAL-TIME CONTROL OF INDUSTRIAL ROBOTS FOR SHAPE SETTING NITINOL RODS

BY

BRADEN EHRAT

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Associate Professor Timothy Bretl

Abstract

Most robots in industrial settings today rely on playback of precise pre-defined coordinates and do not adjust their motion using feedback from sensors. This thesis describes a system to implement real-time motion control of ABB Industrial robots through ROS (Robot Operating System), which enables general use for future experiments that control the robot arms using real-time feedback from force-torque sensors and/or computer vision. Additionally, these robots are used to bend a Nitinol rod into a desired shape and a hardware system has been built to shape-set Nitinol rods with Joule heating.

To my parents, for their love and support.

Acknowledgments

I would like to thank all of the other students in the lab that I have worked with over the past year: Steven Macenski, Avishai Sintov, and Andy Borum. Thank you, Dr. Timothy Bretl, for the opportunity, support, and guidance. Thank you to Konstantin, for our friendship, learning, and adventures during these couple years of school. Thank you to Evelyn, for your affection, understanding, and encouragement. Thank you to my parents, Mark and Lisette Ehrat, who deserve the most credit for where I am today. Finally, I would like to thank the National Science Foundation for grant IIS-1320519, which provided funding support for this project.

Table of Contents

List of Tables vi

List of Figures vii

LIST OF ABBREVIATIONS viii

Chapter 1 Introduction 1

Chapter 2 Background 2
- 2.1 ABB IRB120 Robotic Arms 2
- 2.2 ABB IRC5 Industrial Robot Controller 2
- 2.3 The RAPID programming language 7
- 2.4 RAPID MultiMove 8
- 2.5 Robot Operating System 8
- 2.6 ROS-Industrial 9
- 2.7 The Simple Message protocol 10
- 2.8 A ROS-Industrial Robot Driver Node 10

Chapter 3 Real-Time Control of ABB robots with ROS-Industrial 12
- 3.1 RAPID Streaming ABB Driver 12
- 3.2 Industrial Robot Client 14
- 3.3 Experiments 14
- 3.4 Future Work 17

Chapter 4 A System For Shape-Setting Nitinol Rods With Robotic Arms 18
- 4.1 System Overview 18
- 4.2 Electrical Resistivity 19
- 4.3 Hardware 20
- 4.4 Operator Interface 23
- 4.5 Microchip components 25
- 4.6 Software Control 25
- 4.7 Future Work 26

Chapter 5 Conclusions 27

References 28

Appendix A Hardware Costs 30

List of Tables

2.1 ROS-Industrial current hardware support (Source: [1]) 10
A.1 Approximate Component Cost of Robot-Mounted Nitinol Shape Setter . . . 30

List of Figures

2.1 Two IRB120 robotic arms installed in our lab 3
2.2 Overlapping working ranges of two IRB120 robotic arms 4
2.3 Top and side views of IRB120 working range 4
2.4 ABB IRC5 controller and drive module cabinets 5
2.5 IRC5 Industrial Robot Controller and drive module 5
2.6 IRC5 control module block diagram 6
2.7 IRC5 drive module block diagram 7
2.8 ROS-Industrial ABB downloading driver 11

3.1 Latency between commanded and measured joint angle 15
3.2 Latency between `joint_state` and Accelerometer readings 16

4.1 The shape-setting system end effector 21
4.2 Shape-setting system control box 22
4.3 Shape-setting system remote 23
4.4 Block diagram of the nitinol shaping system 24

LIST OF ABBREVIATIONS

ROS	Robot Operating System
ABB	ASEA Brown Boveri, a Swedish-Swiss maker of the IRB robots
IRB	Industrial Robot series made by ABB
EGM	Externally Guided Motion
IRC5	ABB's fifth generation Robot Controller
TCP	Transmission Control Protocol, a network stream protocol
UDP	User Datagram Protocol, a connectionless datagram protocol
SPI	Serial Peripheral Interface Bus, a serial communication bus
PID	Proportional Integral Derivative, a feedback control loop
ADC	Analog-to-digital converter

Chapter 1

Introduction

This project has been funded by NSF Grant IIS-1320519, the purpose of which is to study the mechanics, manipulation, and perception of deformable objects for robotic manufacturing. This is motivated by capability deficiencies in state-of-the-art industrial robotics today. For example, welding and painting are tasks commonly completed by robots in factories today, often with superior results than human workers. However, these tasks can be pre-programmed to follow precise pre-determined paths, whereas a task such as installing a wiring harness is easier for a human than state-of-the-art robotics to accomplish. The goal of this project is to enable further research into new algorithms for manipulation and perception of deformable objects. To this end, real-time control of an ABB industrial robot via ROS (Robot Operating System) has been implemented. Based of the hardware design described by Gilbert and Webster [2], a system to shape-set Nitinol rods held by robotic arms has been built. The next chapter will provide background into the robot platform and Robot Operating System. Chapter 3 will describe the implementation of real-time control on the ABB IRC5 industrial robot controller using ROS (Robot Operating System). Chapter 4 will describe a system to shape-set Nitinol rods held by the robot arms.

Chapter 2

Background

2.1 ABB IRB120 Robotic Arms

Our lab has two ABB IRB120 robotic arms installed in a shared work area (Figure 2.1), so that they can simultaneously manipulate the same object. The IRB 120 is the smallest robot produced by ABB [3], weighing 25kg. It can be mounted at any angle, and handles a payload up to 3kg. The arm has 6 joints and 6 degrees-of-freedom, with a reach of 580mm. The arms offer precise control and are specified to have .01mm position repeatability.

The two arms are arranged facing each other, so that their ranges of motion overlap (figure 2.2), and the arms can simultaneously grip and manipulate a deformable object for experiments. Figure 2.3 shows the range that each individual arm has from top and side views.

2.2 ABB IRC5 Industrial Robot Controller

Up to 4 ABB robotic arms are controlled by one ABB IRC5 Controller. [4][5] Each robotic arm is connected to a dedicated motor drive that powers the brake release and motors for each of the six joints in each arm, as commanded by the motion planner of the IRC5 controller. In our setup, the IRC5 control module and first arm's drive are located in the same cabinet and the second arm's drive is located in a smaller cabinet, as shown in figure 2.4.

Figure 2.5 shows the inside of the first cabinet, containing the IRC5 control module and the first arm's drive. On the outside, there is an Ethernet network connection for control, programming, and monitoring via computer, and in our case, Robot Operating System. The system has a remote cord connecting to a FlexPendant, a hand-held controller for the IRC5 system. Also on the outside are the Emergency Stop, Motors On button, and the Auto/Hand

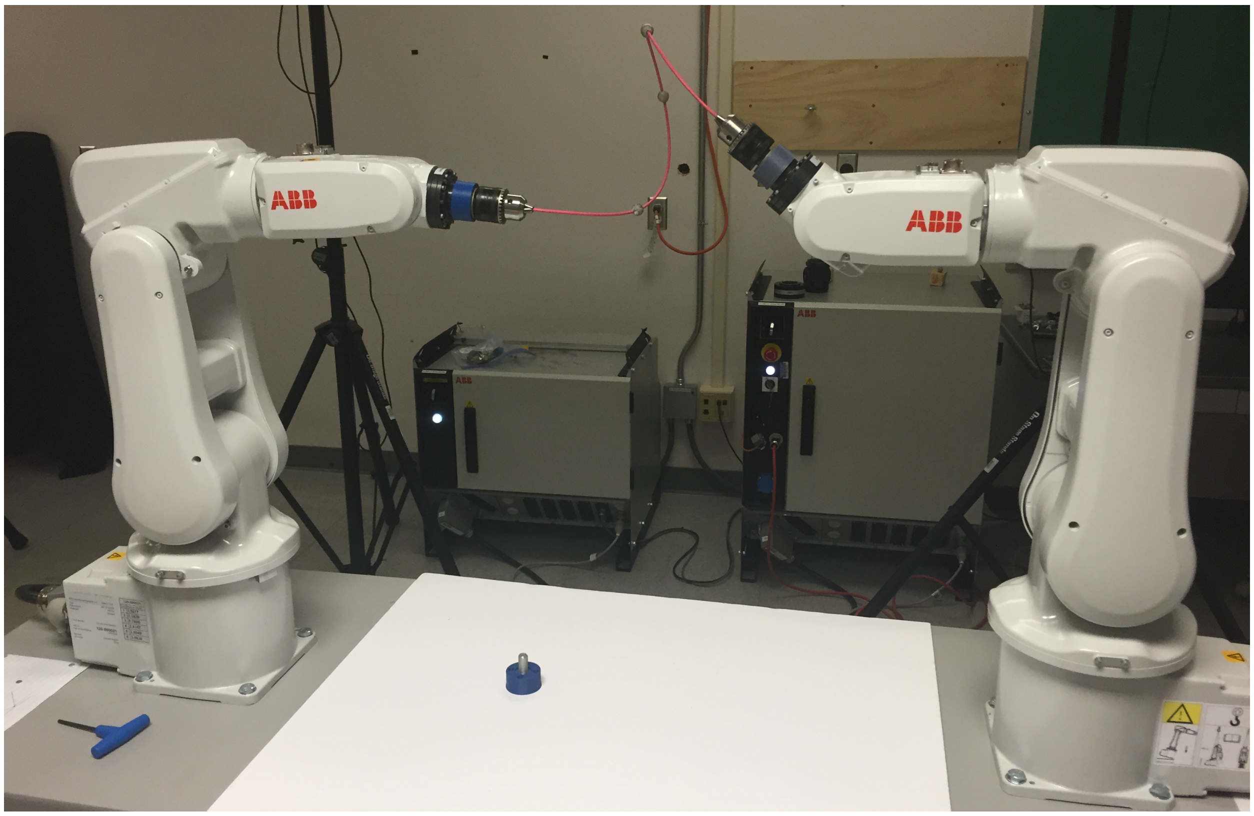

Figure 2.1: The two IRB120 robotic arms installed in our lab. The IRC5 controller is in the right hand cabinet in the background, which also houses the drive module for robot 1. The left cabinet contains the drive module for robot 2.

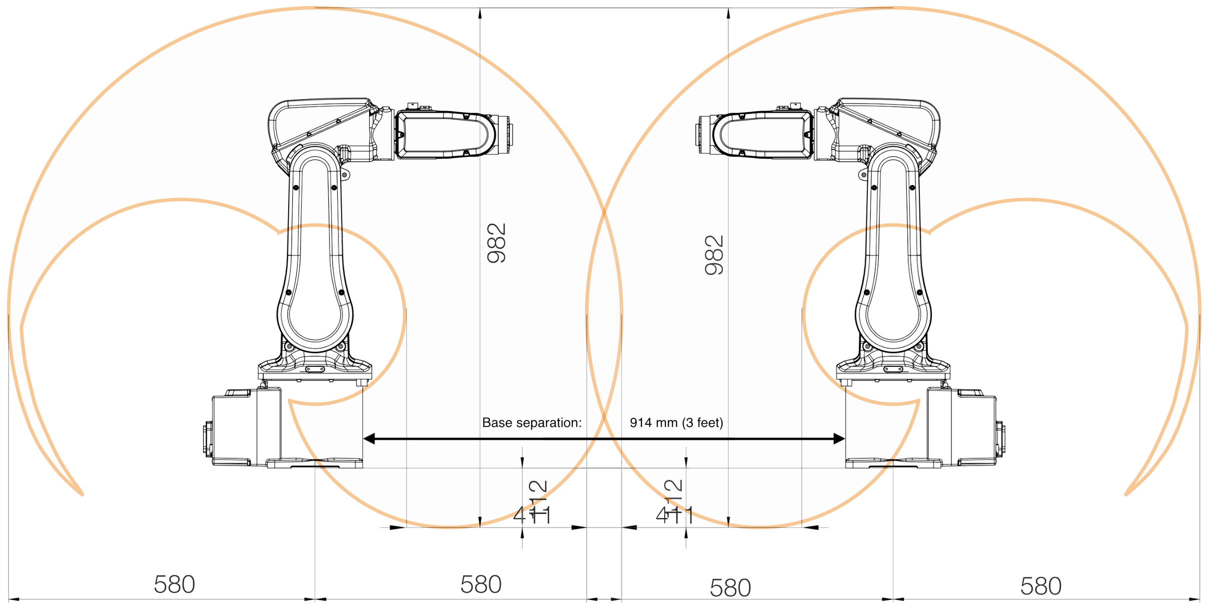

Figure 2.2: Working range of our pair of IRB120 robotic arms installed 3 feet apart (other units: millimeters). The arm's work spaces overlap to allow manipulation of the same object by both arms.

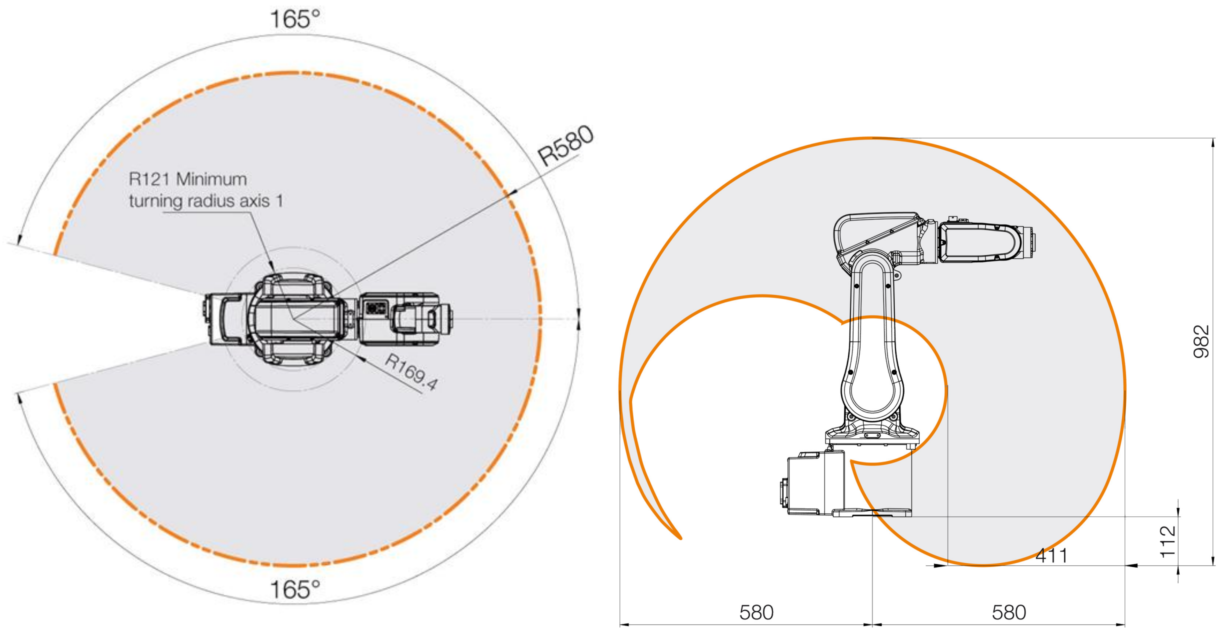

Figure 2.3: Top and side views of one IRB120 robotic arm's working range (units: millimeters).

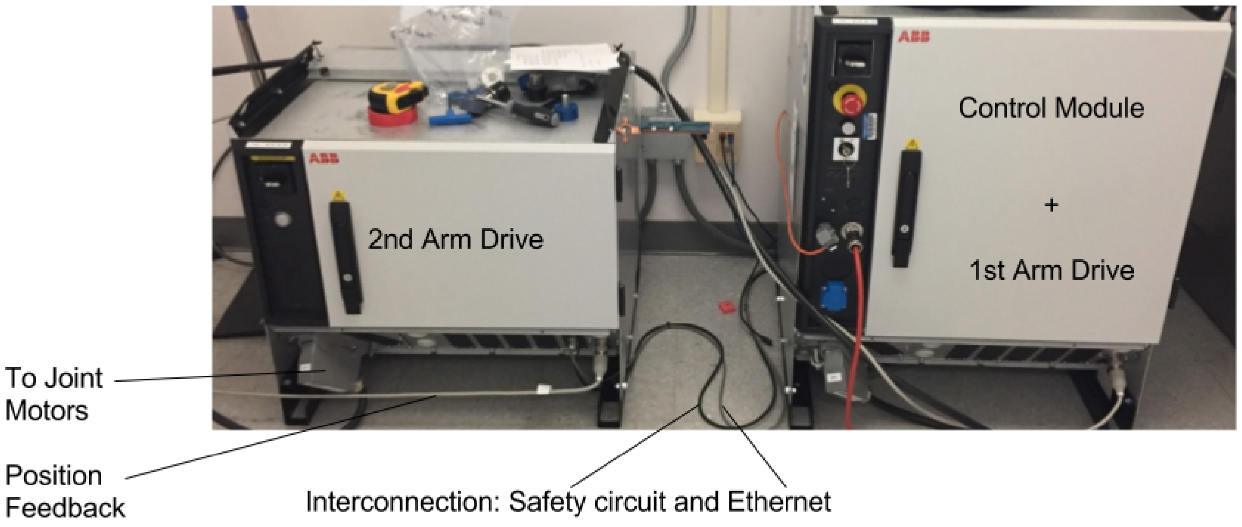

Figure 2.4: ABB IRC5 controller and drive module cabinets

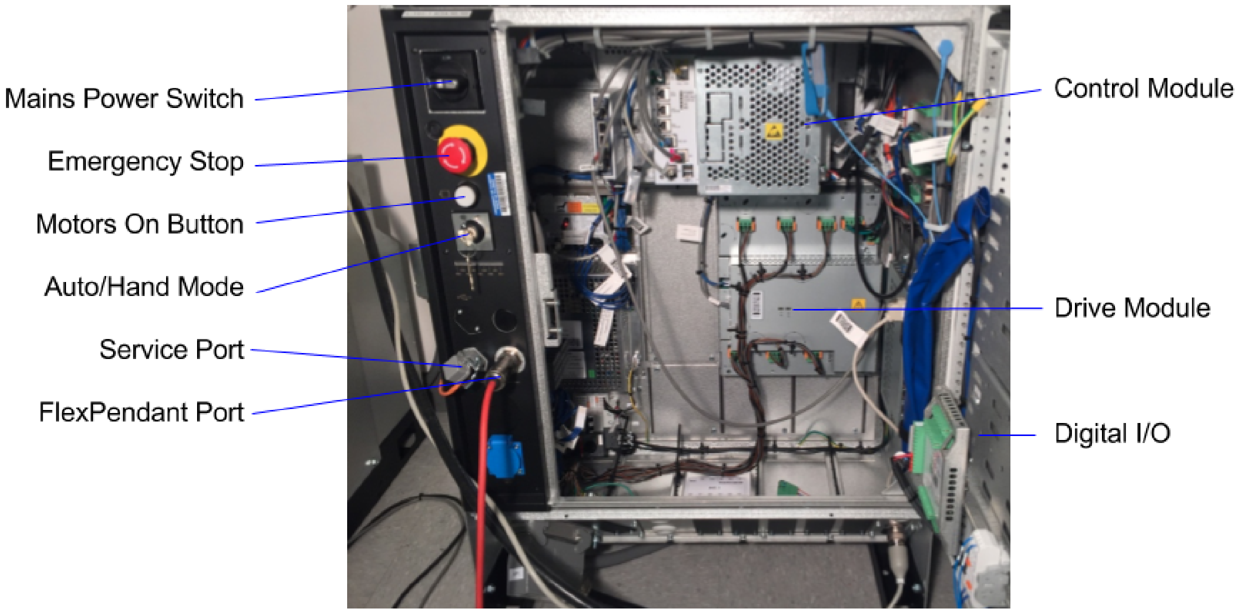

Figure 2.5: IRC5 Industrial Robot Controller and drive module

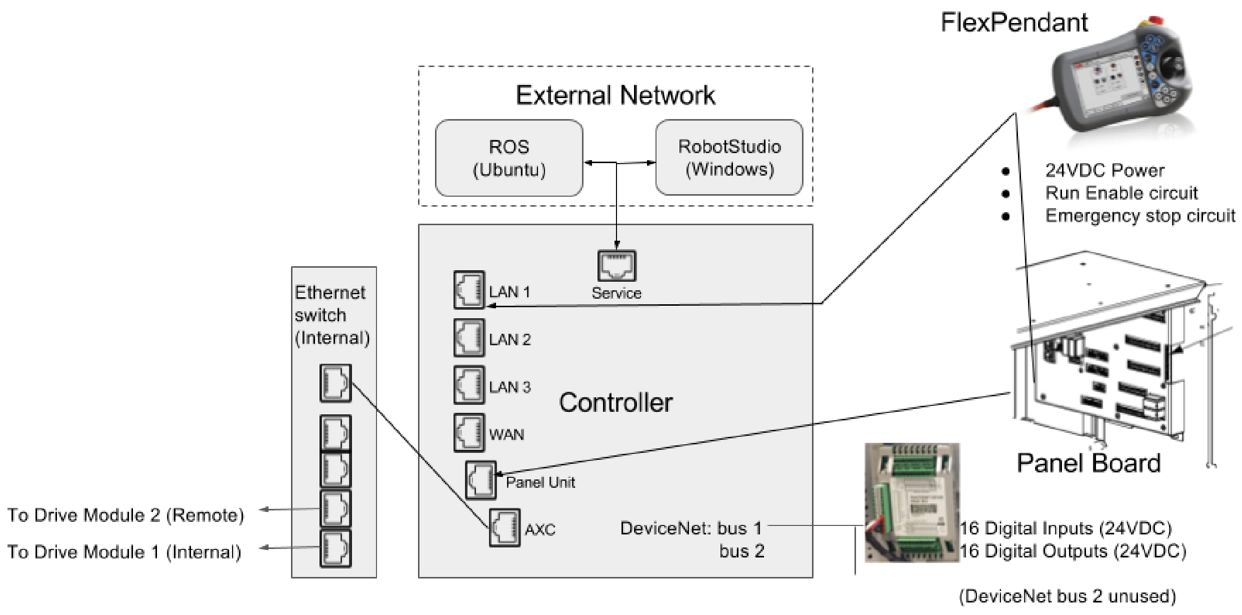

Figure 2.6: IRC5 control module block diagram

mode switch. The Motors On button is pushed to re-enable the drives after startup or a fault. When the Hand/Auto Mode switch is in Hand mode, a dead-mans-switch must be held on the FlexPendant the entire time that the brakes are disengaged and the arms run at a limited speed. When the switch is in Auto mode, the robot will run automatically and up to its full speed. For safety, all experiments so far have been run in Hand mode.

Figure 2.6 shows a block diagram of how the IRC5 controller communicates with other parts of the system. A 5-port industrial Ethernet switch bridges the IRC5 controller and up to 4 drive modules. The Panel Board is a single board for all arms that is connected to the IRC5 controller via Ethernet and has power and safety loop circuitry. The FlexPendant is also part of the physical safety loop through its dead-mans-switch and connects to the IRC5 controller via Ethernet. The IRC5 controller in our lab has a ABB DSQC 652 I/O Unit with 16 digital inputs and 16 digital outputs at 24 volts; it is currently unused. The I/O unit communicates with the IRC5 controller on a DeviceNet bus, to which other units could be added.

Figure 2.7 shows how each arm drive is connected to the rest of the system. In addition to driving an arm's joint motors, each arm drive module measures the current angle of each

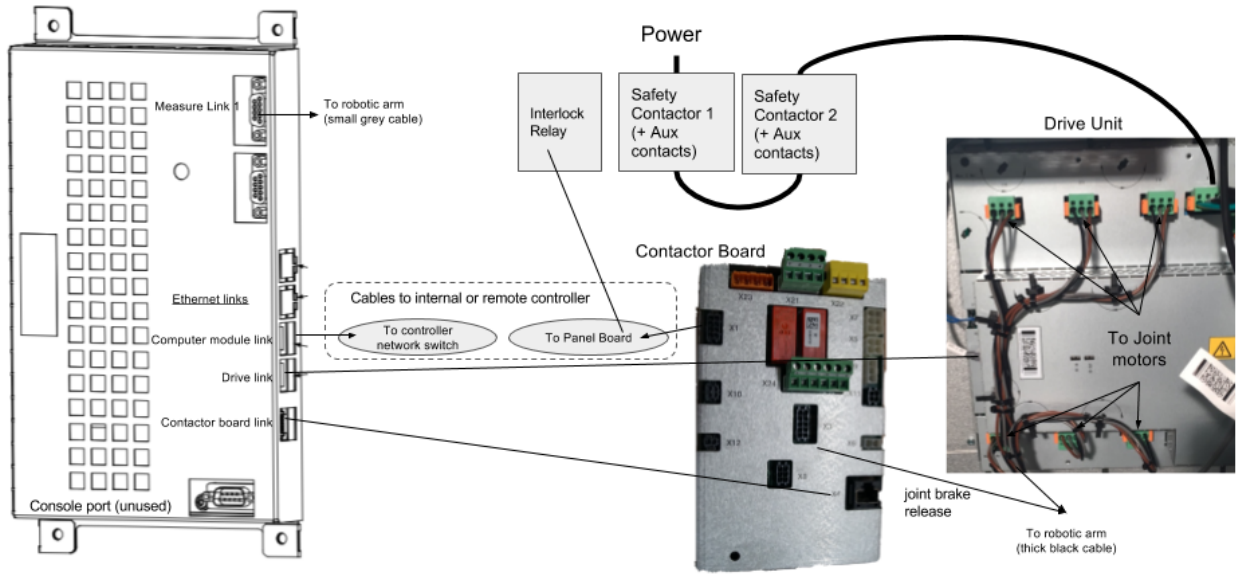
Power
Interlock Relay
Safety Contactor 1 (+ Aux contacts)
Safety Contactor 2 (+ Aux contacts)
Drive Unit
Measure Link 1
To robotic arm (small grey cable)
Ethernet links
Cables to internal or remote controller
To controller network switch
To Panel Board
Contactor Board
Computer module link
Drive link
Contactor board link
To Joint motors
joint brake release
To robotic arm (thick black cable)
Console port (unused)

Figure 2.7: IRC5 drive module block diagram

joint via connections to sensors in the joints; this position feedback data is then returned to the IRC5 controller.

2.3 The RAPID programming language

The IRC5 controller is programmed with the RAPID programming language, ABB's proprietary programming language for its robot systems. It has high-level programming language features, such as functions, procedures, modules, error-handling, trap routines, and multi-tasking that supports up to 10 simultaneous threads.

RAPID code is executed on the IRC5 controller. Robot movements are programmed as pose-to-pose movements and the path between these two positions is automatically calculated by a built-in motion planner. The robot then moves, at the user-desired speed, to reach the next target position. Each pose is in six dimensions and can be specified either as the positions of the six joints or as the 3-D work-space coordinates and 3-D end-effector orientation.

2.4 RAPID MultiMove

The IRC5 controller has built-in support for synchronizing movement between multiple robotic arms driven by the same controller. This feature is broken into two functions, each requiring a separate license:

- MultiMove Independent: the robots work independently at some times and wait for each other at others.
- MultiMove Coordinated: all motion is synchronized for the robotic arms.

The multi-move functionality is used through special RAPID instructions. For MultiMove Independent, the `WaitSyncTask` instruction can be used to synchronize multiple processes at a point. Each process calls the `WaitSyncTask` instruction with a special variable that other tasks wishing to coordinate with it use as well. Once all tasks reach their `WaitSyncTask`, each task starts simultaneously, but otherwise runs independently from then on.

In normal operation and in MultiMove Coordinated, there is a motion planner for every motion task that executes the movement instructions for its task. When using MultiMove Coordinated, however, every task wishing to have coordinated motion uses the same motion planner. Every task wishing to coordinate motion first calls `SyncMoveOn`, executes the movements (each indexed with an identifier corresponding to the same movement in different tasks) and then all tasks call `SyncMoveOff`. With MutliMove Coordinated, all coordinated movements start and end at the same time.

2.5 Robot Operating System

Robot Operating System (ROS) is a framework for controlling robots and is most often used in research and academic settings. It is not what is normally called an ”operating system”, since it does not manage lower-level hardware and resources directly. Rather, it provides software frameworks and acts as a middle layer for networked communication between robotics hardware and control software [6]. ROS uses a publish-subscribe pattern for inter-process communication over channels called “topics”.

2.6 ROS-Industrial

ROS-Industrial is an extension project that aims to extend the features of ROS to support robots in industrial settings [7]. ROS-Industrial provides a common control interface for industrial robots from different manufacturers so generic motion control software can control them. Currently, robotic arm controllers made by ABB, Adept, Fanuc, Motoman, and Universal Robot are supported. A running ROS-Industrial setup creates a ROS node for the industrial robot that has a manufacture-independent interface for controlling the robot's motion and getting feedback of the robot's current joint states.

ROS-Industrial supports a few distinct modes of operation for communicating motion instructions to industrial robots.

- **Trajectory Downloading**: A series of joint positions is sent to the robot and the robot executes the series when all joint trajectory points have been downloaded to the robot controller. The trajectory includes a velocity for each position which the robot controller adheres to. After executing a trajectory, the robot stops and waits to download the next trajectory.
- **Trajectory Streaming**: Point positions are streamed to the controller similarly to Trajectory Downloading, but the robot executes the commands as it receives them, possibly subject to a small buffer.
- **Position Streaming**: Joint positions are streamed to the robot controller from ROS-Industrial, but the robot velocity is fixed by the robot controller.

In the current table of supported hardware provided by ROS-Industrial, only one of the three modes is implemented for any robot:

The asterisk(*) denotes that this thesis describes an implementation of Trajectory Streaming for ABB robots controlled by the IRC5 controller.

Table 2.1: ROS-Industrial current hardware support (Source: [1])

Vendor	Controller(s)	Position Streaming	Trajectory Downloading	Trajectory Streaming
ABB	IRC5	NO	YES	NO*
Adept	CX, CS	YES	NO	NO
Fanuc	R-30iA	YES	NO	NO
Motoman	DX100	NO	NO	YES
Universal Robot	UR 5	YES	NO	NO

2.7 The Simple Message protocol

In any mode, the ROS-Industrial node communicates with the controller using the Simple Message protocol. The Simple Message protocol is a binary protocol that usually runs over TCP sockets, but could use serial or UDP channels as well. It is standardized so that most robot hardware can use the generic Industrial Robot Client of ROS-Industrial. However, if customization is required to support particular hardware, it is desired that it be added at this level instead of changing the interface provided by ROS topic messages, so external software is still provided the same interface.

2.8 A ROS-Industrial Robot Driver Node

According to the ROS-Industrial Robot Driver Specification [8], a motion control node should accept commands from two ROS topics: `joint_path_command` and `joint_command`

- `joint_path_command`: ROS-Topic with JointTrajectory messages. Each message contains a series of joint positions and velocities/durations that will executed by the robot.
- `joint_command`: ROS-Topic with JointTrajectoryPoint messages. One joint position for the robot to execute. This should allow dynamic streaming of points for real-time control.

However, upon inspection of all current ROS-Industrial code, a subscription to the `joint_command` topic is not implemented by any generic or manufacturer-specific robot drivers. Indeed, only a Github issue was found to note the missing functionality [9]. The current generic robot

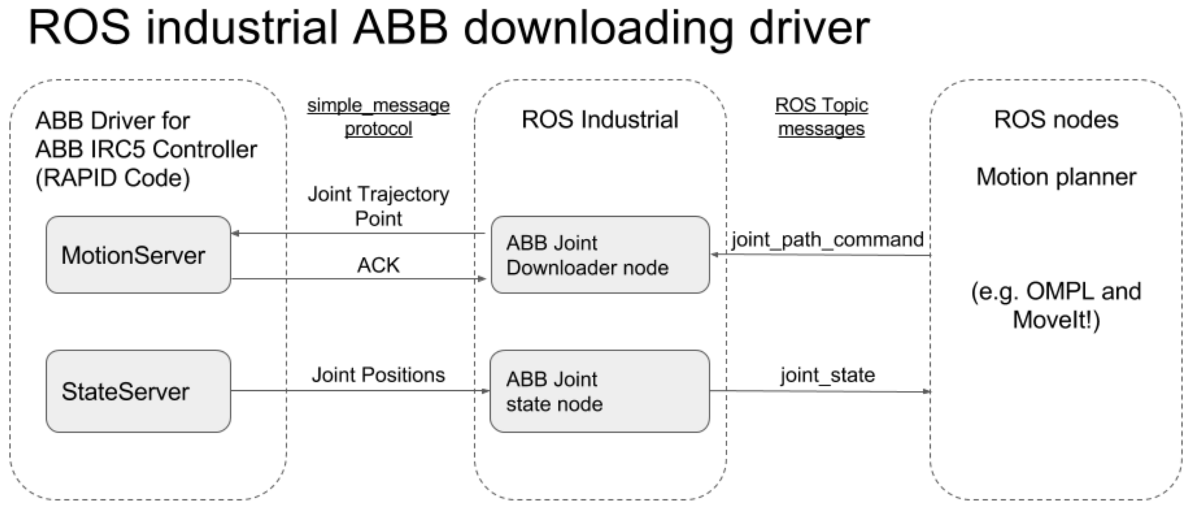

Figure 2.8: ROS-Industrial ABB downloading driver

drivers do implement Trajectory Streaming, but they only do so from JointTrajectory messages. If the JointTrajectory message contains only one point, the generic robot client will actually send the robot controller the same point twice, with the first set to start motion and second to stop motion. In addition, there is a 250 millisecond polling period in the generic streaming client while waiting for new JointTrajectories to stream. It is evident that the ROS-Industrial framework does not support real-time control with any supported hardware today.

Figure 2.8 shows a system block diagram containing the ABB IRC5 Robot control, the ROS-Industrial node connected to it, and other ROS nodes that control and get feedback from the robot through ROS message topics. The next chapter will describe work improving this system to allow real-time control of ABB robots.

Chapter 3

Real-Time Control of ABB robots with ROS-Industrial

It was desired that the ABB robots in our lab are controllable in real-time. As described in the background chapter, ROS-Industrial currently only supports Trajectory Downloading for ABB robots, thus, it was necessary to implement Trajectory Streaming on the ABB robots. In addition, a ROS node was implemented to subscribe to JointTrajectoryPoint messages from the `joint_command` topic and stream them to the robot controller.

3.1 RAPID Streaming ABB Driver

To implement Trajectory Streaming in RAPID code on the ABB IRC5 controller, the network server and motion tasks were modified to only consider the current, most recent point received. The motion server listens on a TCP port for a client connection from the industrial robot client ROS node. A motion task then runs for every robot arm that is to be simultaneously controlled. The motion task is written to dynamically determine its Mechanical Unit Index, so that the same source code file does not need to be copied or edited to run multiple motion tasks. Configuring the arms to be controlled on the IRC5 controller only requires that the motion code is set to run as a task for every mechanical unit and the motion task be added to a configuration list. The MultiMove Independent features are utilized to synchronize the motion-server task with all of the motion tasks. The `WaitSyncTask` function was employed to block the motion tasks until a joint command was received by the motion server.

Listings 3.1, and 3.2 shows selected lines of the source code for the server and motion tasks, respectively. When the motion server receives a Trajectory Point message, it puts the next joint target and duration into variables shared by the motion server and all motion tasks. It then calls `WaitSyncTask`, which allows the motion tasks to continue and to read the next target. Then, `WaitSyncTask` is called a second time by both the server and motion tasks. In this use, `WaitSyncTask` has two functions: protecting read and write access to the shared variables and synchronizing the motion of multiple arms. If a license for Coordinated MultiMove were obtained, `SyncMoveOn` could additionally be used to more closely synchronize movement of multiple arms through the same motion planner.

Listing 3.1: Selected lines from the real-time trajectory driver's server task

```
PROC main()
    WHILE TRUE DO
        ROS_receive_custom_joints client_socket;
        WaitSyncTask ROS_sync1, \Timeout:=.5;
        ! Motion servers copy shared variables here
        WaitSyncTask ROS_sync2, \Timeout:=.05;
        ! Send reply after motion accepted so
        ! client can implement rate control
        ROS_send_msg client_socket, reply_msg;
    ENDWHILE
ENDPROC
```

Listing 3.2: Selected lines from the real-time trajectory driver's motion task

```
PROC main()
    VAR jointtarget target;
    WHILE TRUE DO
        WaitSynctask ROS_sync1, ROS_sync_list;
        ! Copy data from shared variables
        target.robax := ROS_joints{mecunit_idx};
        MoveDuration := ROS_duration;
        WaitSynctask ROS_sync2, ROS_sync_list;
        MoveAbsJ target, move_speed, \T:=MoveDuration,
                stop_mode, tool0;
    ENDWHILE
ENDPROC
```

3.2 Industrial Robot Client

Although there exists generic implementations of industrial robot clients for ROS-Industrial node, it was determined that the current implementation of both Trajectory Streaming and Trajectory Downloading were ill-suited for real-time feedback control, due to limitations discussed in the background. A new ROS node was created that subscribes to the `joint_command` ROS topic and accepts single JointTrajectoryPoint messages that it then forwards to the robot controller.

The standard SimpleMessage protocol has a fixed-size payload and carries 10 joint angles, regardless of the number of angles the robot actually has. Since our two robot arms have 12 joints combined, this implementation re-compiled the ROS-Industrial library with the number of joints in the Simple Message increased from 10 to 12. The motion server RAPID code was also hard-coded to use this new packet size.

3.3 Experiments

The first experiment aims to measure the delay between commanding the robot to move to a position and when the system reports that the joints have moved to that position. A program was written to publish joint commands at 30 Hz that commanded a single joint of one robot arm to angles that followed a sinusoidal motion over a range of $\pi/8$ radians with a period of 2 seconds. A logging program listened on the ROS topics `joint_path_command` and `joint_states`, logging each joint command and state message received to CSV files with each line prepended by a global timestamp. $T = 0$ was set to the first point timestamp from any source.

Using `scipy.optimize.leastsq`, a least-squares fit was made to each data series to recover the measured amplitude, period, and phase of the sine wave. The control latency is

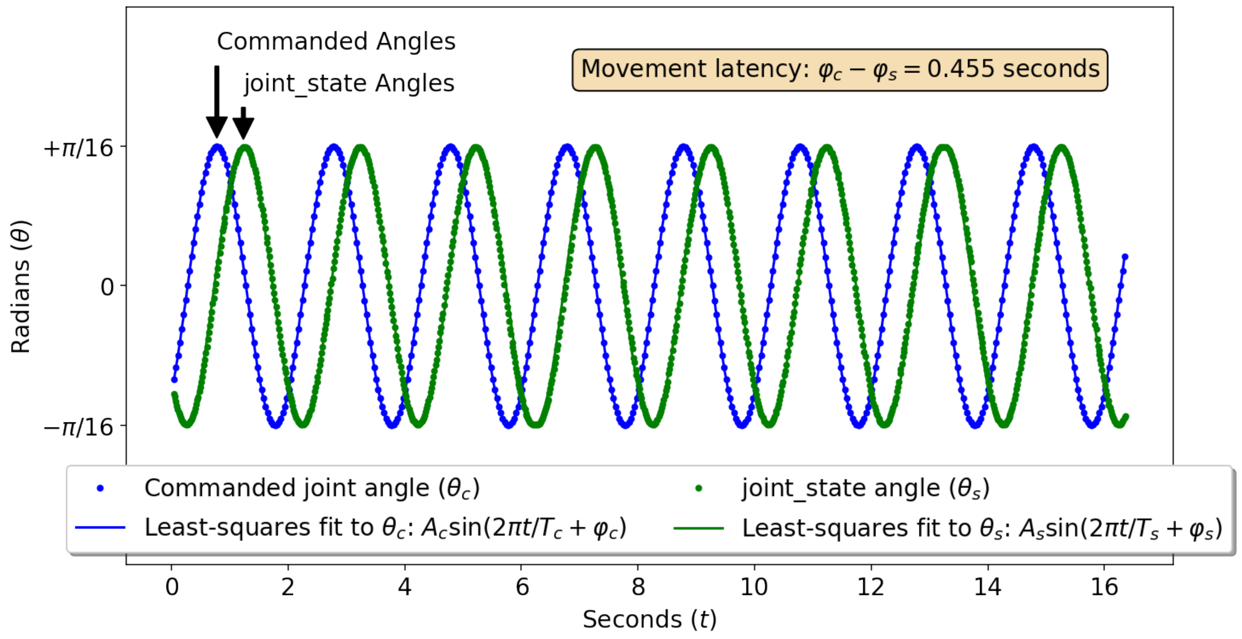

Figure 3.1: Latency between commanded and measured joint angle

then the phase difference between the sine wave of the commands and that recorded by the joint angle, to a higher precision than would otherwise be possible with the sample rate.

Figure 3.1 shows each point measured and the fitted curves for the command and measurement signals. The fitted sinusoidal waves for the commanded and measured movements are 455 milliseconds out-of-phase. It would be expected that the measured positions lag by the period of the commanded positions, which is 33 milliseconds. The remaining $455 - 33 = 422$ milliseconds of latency are due to some other delay between sending joint commands and the movement being registered from the measured angles. It is thought that all of this delay is due to the motion-planner on the IRC5 controller. Local experiments were also done in RAPID code to confirm that the motion-planner introduces delays in the hundreds of milliseconds, even if tens of move instructions have been executed before the physical drives respond to the first commands.

While measuring the latency between commanded and measured joint angles, it is unknown what portion of the delay is introduced before the robot arm begins to physically move for commands, or if any delay is introduced in the measurement and communication of the joint states through the RAPID program and back to the ROS node. To attempt

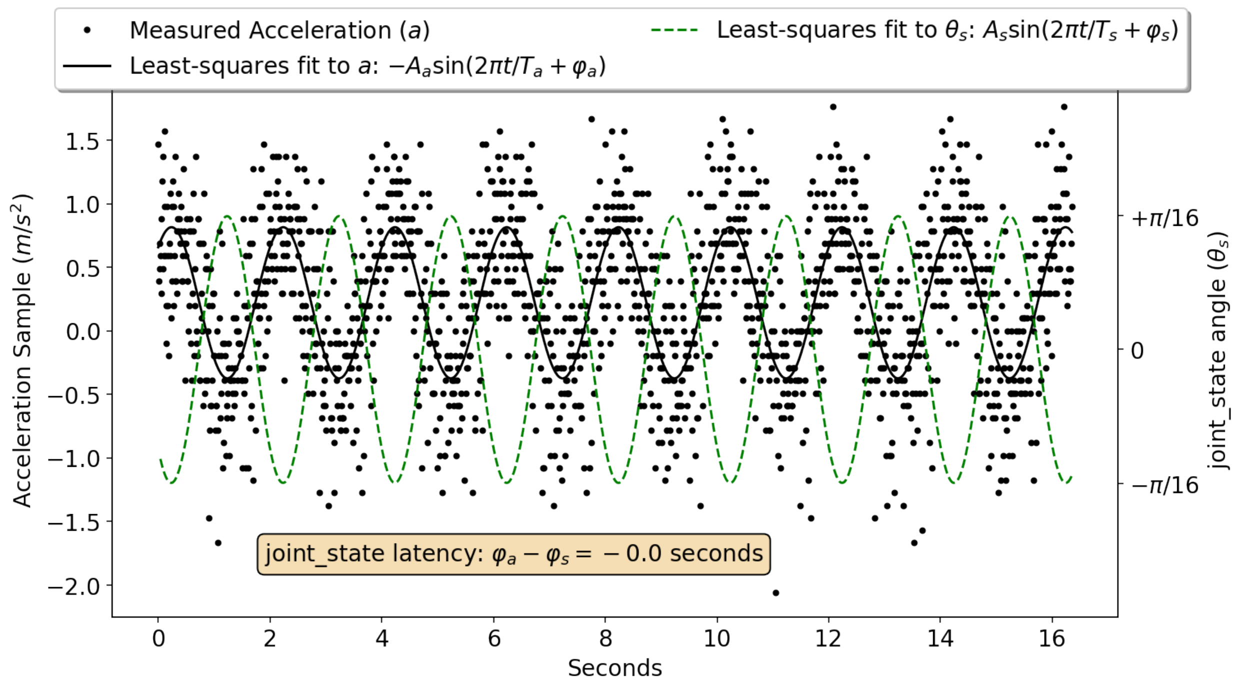

Figure 3.2: Latency between `joint_state` and Accelerometer readings

a more direct measurement of the physical arm movement, an accelerometer was attached to the robot arm and this data was logged with the same program that was recording the commanded and measured joint angles.

Since the joint positions are commanded and measured by the robot by $sin(t)$, the expected acceleration is the second derivative: $-sin(t)$. Figure 3.2 shows the accelerometer measurements over time as the robot was moving as commanded in the previous trial, with the fitted curve for angles measured by the robot controller overlaid as a dashed line.

Although the accelerometer readings were noisy, a least-squares fit was still able to be found, and the measured frequency matched the 2 Hz of the commanded motion. In this trial, the `joint_state` signal was found to have -5 milliseconds of latency from the accelerometer readings, meaning that the accelerometer readings could be delayed. However, the delay was measured in two additional trials as -1 and $+12$ milliseconds. While this a relatively noisy and inaccurate measurement of the `joint_state` latency, it does show that practically all of the latency in the real-time control loop is due to the motion planner.

3.4 Future Work

To allow this work to be used by the broader community, it would be desirable for the solution to be made more generalized. The Simple Message protocol could be extended to support variable numbers of joints, so that ROS-Industrial can work generically with multiple-arm robots from different manufacturers. The generic ROS-Industrial robot client nodes could receive JointTrajectoryPoint messages from the `joint_comand` topic to support real-time control of all robots for which ROS-Industrial has a trajectory streaming driver.

Further work could be done specifically for ABB robots to enable lower-latency control. A new feature called Externally Guided Motion (EGM) was added in ABB RobotWare 6.0, released in 2015. EGM bypasses the path planner and gives direct control to the motor reference generator. A separate license purchase is required to use EGM [10], and it was unable to be tested in this work. Ørjan [11] implemented a system to use EGM to follow a disc rolling down a ramp and tracked by a camera. With EGM, the robot path can be updated every 4 milliseconds (250 Hz), with a control lag of 8 to 20 milliseconds [12].

Chapter 4

A System For Shape-Setting Nitinol Rods With Robotic Arms

This chapter describes a system for shape-setting nitinol rods that are held into a desired shape by a pair of robotic arms. “Nitinol” is the short name for a metal alloy of nickel and titanium, usually consisting of 55-60% nickel, and is a shape-memory alloy with superelastic properties. Its superelastic properties mean that it can be deformed to very high strains (up to 8% strain), and still return to its original shape without heating (like a spring). Since the “superelastic” properties are caused by a phase transformation due to the stress of bending, a more accurate name is psuedoelasticity. This superelastic property is utilized in many applications that require substantial flexibility and motion of metal without deformation, such as concentric tube robots. As a shape-memory alloy, it can be deformed and will return to its original shape when heated. The original shape is also called the ”parent” shape. The parent shape can be set by heating the nitinol to even higher temperatures, around 500 °C, a process called *shape setting* [13].

4.1 System Overview

With our two robotic arms, we will hold a nitinol rod in a desired shape and then heat it to set the shape. If the two end-effectors that grip the rod are connected to a power supply, the rod can be heated via joule heating by passing an electric current though it. Gilbert and Webster [2] have built such a system for shape-setting nitinol tubes and rods, which modulates an electric current through the rod and measures its resistance as a proxy for its temperature. The system is relatively inexpensive, rapid, and accurate, with a mean absolute temperature error of 10 °C from the target temperature.

Gilbert and Webster improve upon other shape setting methods, such as furnace-based approaches, and have more accurate results. However, a drawback of Gilbert and Webster’s shape-setting system is that it requires a jig to hold the nitinol rod in the desired shape during heating. A jig must be custom crafted for any shape desired, which takes time to construct and also reduces the quality of parts as heat transfers through contact points. Our system has the advantage that no construction process is required for custom shapes and the robots hold the rod in the air, providing uniform heat transfer.

During heating, the current and voltage across the part is measured and the part's resistance can be directly measured by application of Ohm's Law; the temperature rise can be calculated by proxy to the rise in the rods electrical resistance. With our robotic arms, the desired shape of the rod could simply be set in software and the robots will quickly bend the rod into the desired shape. The shape-setting system will heat up the rod as the robots hold its ends, eliminating any contact points except the ends. This system could enable rapid prototyping of customized parts for things like concentric-tube robots, and may produce superior quality parts.

4.2 Electrical Resistivity

Electrical resistivity is an intrinsic property of a material that quantifies its opposition to electrical current. The resistance (R) of a rod is then a function of its length (l), cross section (A), and resistivity (ρ):

$$R = \rho \frac{l}{A} \tag{4.1}$$

The temperature coefficient of resistivity quantifies the resistivity change in a material given a temperature change. Often, a linear approximation of the temperature coefficient is used. The temperature coefficient of materials is often given at 20 °C, thus the resistance of the rod at a temperature (T), with a known 20 °C resistivity (ρ_{20}), and temperature coefficient of resistivity (α), can be calculated:

$$R = (\rho_{20} + \alpha(T - 20))\frac{l}{A} \tag{4.2}$$

In our shape setting system, the goal is to reach and hold a target temperature for an amount of time. Since the system measures the rod's resistance, the above formula could be used to directly determine the target resistance if the length and width of the sample are known. However, the length and width dimensions of each sample may be hard to measure accurately enough for a rod to be shape-set. A more uniform measure of each part to be shape set is the starting temperature, which may be simply taken as the temperature of the surrounding room. If the ratio of the target final resistance (R_F) to the initial resistance (R_I) is made using using equation 4.2, the length and cross-section parameters cancel, leaving only the resistivity and temperature parameters:

$$\frac{R_F}{R_I} = \frac{\rho_{20} + \alpha(T_F - 20)}{\rho_{20} + \alpha(T_I - 20)} = \frac{\rho_F}{\rho_I} \tag{4.3}$$

Using equation 4.3, the only input parameter the shaping controller requires is the expected ratio of the initial to target resistivity. The system will measure the initial resistance of a particular rod when the shaping process is first started and then calculate the target resistance. This method improves the potential productivity of the system for rapid prototyping, as the system can quickly perform the same temperature setting on multiple parts of different dimensions without changing any input parameters.

4.3 Hardware

Our robots currently have a drill chuck mounted on each arm, which can be tightened to grip a rod. The drill chucks are good gripping devices, since they hold the rod in a precise location relative to the robot. They also serve the shape-setting system well, as the chucks will conduct electricity to the rod.

However, it is required that the robot arms are shielded from the voltage and heat of the shape-setting process. A ceramic bolt was used to rigidly hold the end-effector to the robot, chosen because ceramic is a great electrical and thermal insulator. An image of our shape-setting end-effector is shown in figure 4.1. The wire lead from power source is attached to the drill chuck with a 1/2 inch bolt. The bolt has 5/16 inch threads tapped into its face, which the ceramic bolt screws into. A 3-D printed adaptor holds the ceramic bolt to the

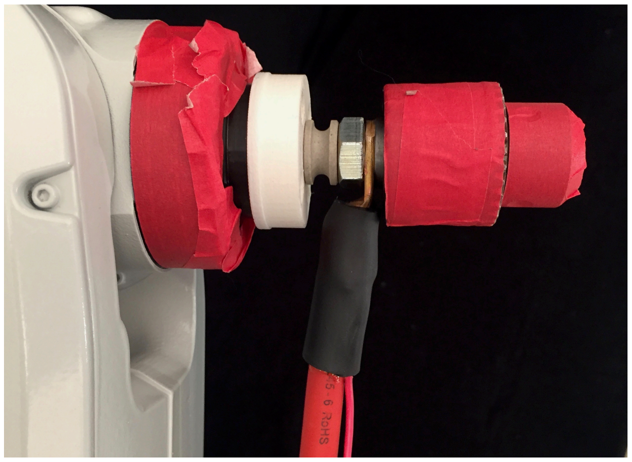
Figure 4.1: The shape-setting system end effector

robotic arm and a ceramic washer separates the adaptor from the chuck.

The design of the system's control circuit board is based on Gilbert and Webster's and we thank them for providing their source code and schematics. However, our system is designed for a higher level of safety and durability; the enclosure, operator-interface, high-power components, and safety design/interlocks are designed to industrial standards. The MOSFET, current-measurement shunt, Arduino, and associated control circuitry is installed in a fiberglass electrical enclosure, as shown in figure 4.2. The enclosure is 12x10x6 inches and has an aluminum subpanel inside, onto which the other components can be mounted. The system has a remote control on a 50 foot cable, which has a key switch and a button that must be held to enable the electromechanical relays inside the control box in manual mode, unless it's allowed to be enabled by a relay connected to the robot's safety circuit. These buttons/relays enable a 500-amp relay in series with the MOSFET.

The system power source for shape-setting is a standard 12-volt (6-cell) lead-acid spill-proof car battery, rated at 1000 cranking amperes [14]. The battery's charge is maintained by a dedicated charger that also powers the control circuit. Additionally, the battery is connected via a 300-amp fuse to protect against overload from a connection or software error.

Besides the battery and charger, all other components are mounted inside an electrical enclosure to increase safety and sturdiness. The enclosure can be mounted under the robots and has indicators to show the current status. A chassis-mounted MOSFET was chosen,

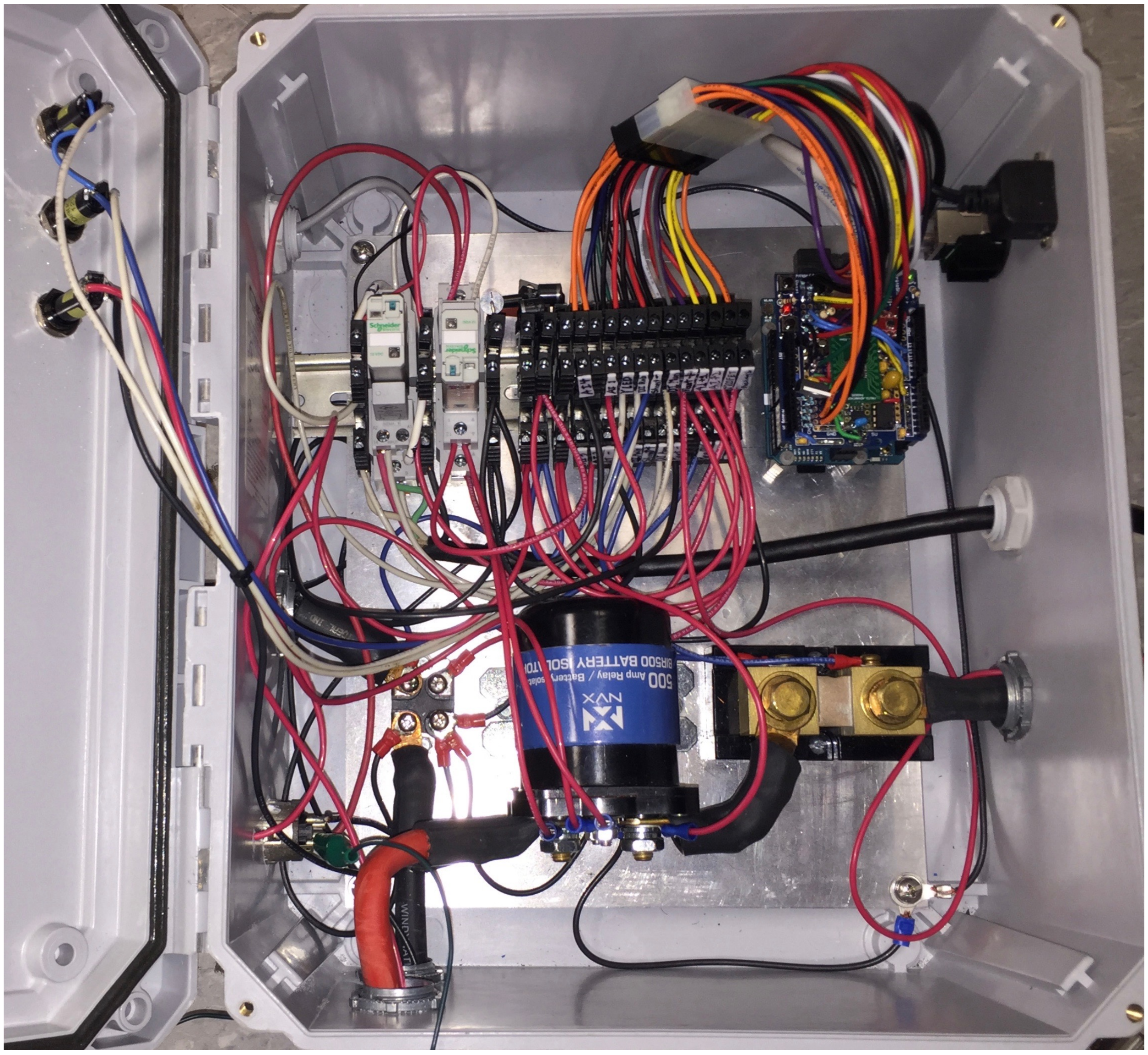

Figure 4.2: Shape-setting system control box

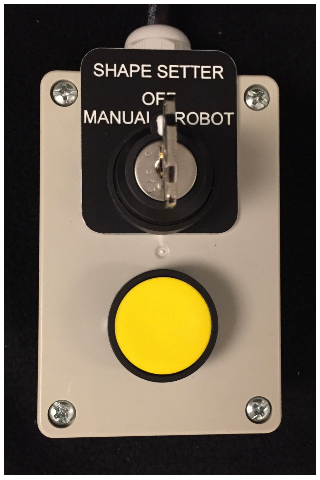

Figure 4.3: Shape-setting system remote

which offers high-speed switching to regulate the power delivered to the part. The IXYS IXTN660N04T4 MOSFET we use is rated for 200 ampere continuous and 1800 ampere instantaneous current [15]. For measuring the current flowing through the nitinol wire, a chassis-mounted shunt was placed in series with the circuit through the nitinol rod. The shunt is rated up to 1000 ampere, and is also chassis-mount.

Local control of the shape-setter is provided by an Arduino microcontroller. The Arduino can quickly switch the MOSFET so that the average power delivered to the part can be regulated. The Arduino measures feedback by the voltage and current flowing through the nitinol part to control the part's temperature. The final cost of our shape-setting system was less than $1000 (breakdown in Appendix A).

4.4 Operator Interface

The Nitinol shaping system has a 50-foot corded remote control with a selector switch and momentary push-button, shown in figure 4.3. The selector switch requires a key to operate and has three positions:

- **Auto** – The shape-setter is interlocked with the robot's safety circuit, so that the shape-setter will be enabled simultaneously when the robot is engaged. If the robot is in its Hand mode, the operator must hold the FlexPendant dead-man-switch.

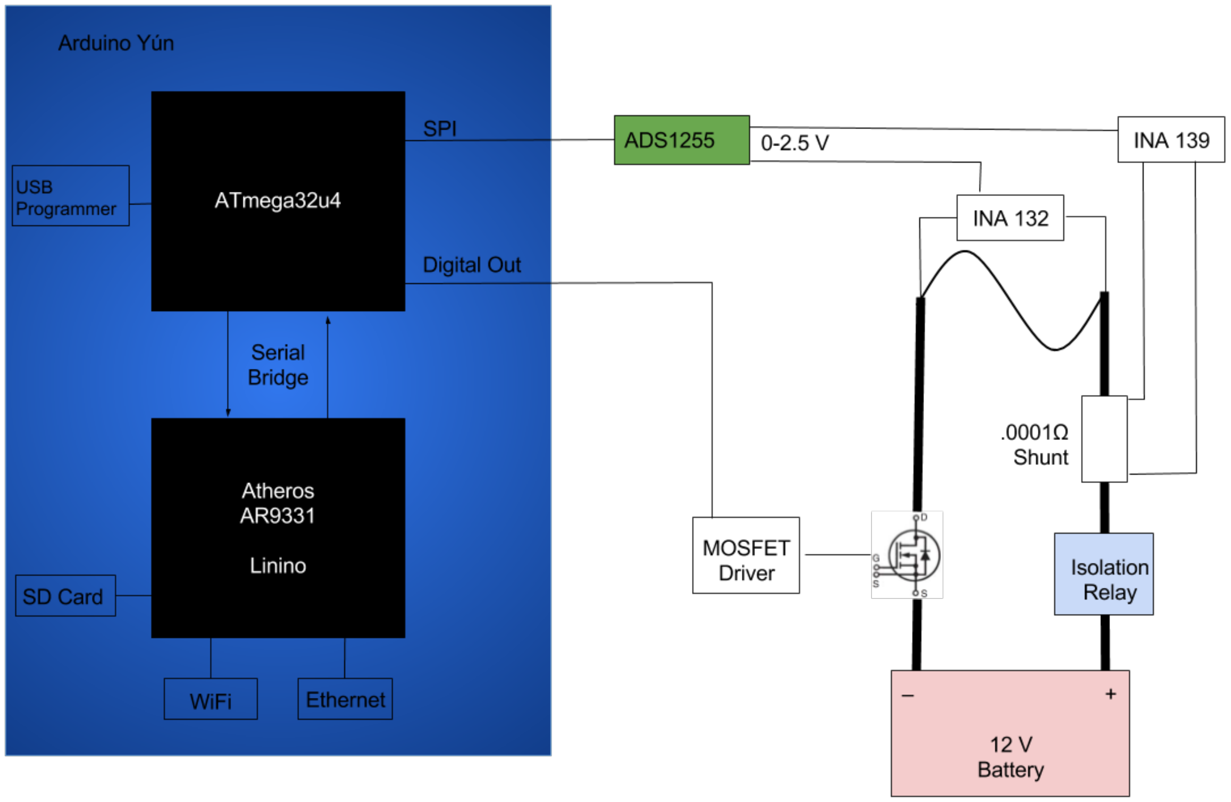

Figure 4.4: Block diagram of the nitinol shaping system

- **Manual** – The shape-setter will work without the robot, but the push button on the remote must be held for the duration of shape setting.

- **Off** – Disables the shape-setter. The key can be removed to prevent unauthorized operation.

There are three LED indicators on the front panel of the control box: green, yellow, and red. The green and yellow lights are controlled by the Arduino. During normal operation, the green LED will light when powered and the yellow will light when the shape-setter is "armed". The red LED is tied to the main relay and indicates when the shape-setting system is engaged.

4.5 Microchip components

The Arduino Yún was chosen for this project [16]. Figure 4.4 shows how the various microchip components are connected. It has the ATmega32U4 microcontroller, a standard AVR Arduino microcontroller, the same as that used in the Arduino Leonardo, and has the the standard Arduino headers for expansion boards. The microcontroller is tasked with controlling the rod heater feedback loop. Unlike other Arduino boards, the Yún also has a second microprocessor, the Atheros AR9331, to which Ethernet, Wi-Fi, and a Micro-SD card reader are attached. The Atheros processor runs Linux at a clock speed of 400Hz. The Linux system is tasked with high-level functions, like communicating with ROS or a web client for input parameters and reporting the system status.

The voltage and current of the system are measured through an analog-to-digital conversion, but the Arduino's built-in analog-to-digital converters are too slow and have insufficient of resolution for this system. Instead, an external analog-to-digital converter, the ADS1255 from Texas Instruments, was used to provide 24-bit resolution at up to 30,000 samples-per-second [17]. It is connected via the Arduino's SPI bus and a client library for the Arduino was written to communicate with it. An INA132 [18] difference amplifier is used to measure the relative voltage across the part and output a 0-2.5 volt signal that the ADS1255 measures relative to ground. For measuring the current, an INA139 [19] converts the voltage across the current-measurement shunt into the second channel for the ADS1255 to measure.

4.6 Software Control

The software has both safety and control responsibilities. For safety, the Arduino's low-resolution analog inputs are connected to various points in the system to ensure that it is operating nominally and will stop the process if any error is detected. The AnalogScanner library [20] was used to read the analog inputs via interrupts, so the main control loop does not block while waiting for the ADC to perform conversions.

While shape-setting, a PID controller is used to regulate the rod's temperature. The input to the PID controller is the error between the rod's target resistance and its measured resistance and the output is the amperes to be delivered to the rod. Since instantaneous current to the rod can only turned on or off with the MOSFET, an exponential moving average with a time constant of 10 milliseconds is used to track the average amperage delivered. A simple boolean expression is then evaluated every control loop to turn the MOSFET on if the averaged current is less than the target current from the PID controller.

4.7 Future Work

Experiments should be performed to determine the temperature accuracy and effectiveness of the shape-setting system. Additionally, programs could use the shape-setting system and the robots together by controlling both the robot arms and shape-setting process through ROS.

Chapter 5

Conclusions

In this work, real-time control of ABB robots has been implemented through a modified ROS-Industrial installation. It is limited, however, by a 400-500 millisecond delay that the ABB IRC5 robot controller introduces with its motion planner. Future work may use the Externally Guided Motion feature to dramatically lower the control latency. Finally, a system for shape-setting nitinol rods held in shapes with the robot arms has been implemented.

References

[1] "ROS-Industrial Supported Hardware," April 2017. [Online]. Available: https://wiki.ros.org/Industrial/supported_hardware

[2] H. B. Gilbert and R. J. Webster III, "Rapid, reliable shape setting of superelastic nitinol for prototyping robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 98–105, 2016.

[3] *IRB 120 data sheet*, November 2016. [Online]. Available: https://library.e.abb.com/public/3bd625bab3c7cae1c1257a0800495fac/ROB0149EN_D_LR.pdf

[4] "IRC5," November 2016. [Online]. Available: http://new.abb.com/products/robotics/controllers/irc5

[5] *IRC5 Industrial Robot Controller data sheet*, ABB, November 2016. [Online]. Available: https://library.e.abb.com/public/bedd1769ea1e4bb9c1257da10037e215/IRC5_IndustrialRobotController_ROB0295EN.pdf

[6] "Robot Operating System," December 2017. [Online]. Available: http://www.ros.org

[7] "ROS-Industrial," December 2017. [Online]. Available: https://rosindustrial.org

[8] "ROS-Industrial Robot Driver Specification," December 2017. [Online]. Available: https://wiki.ros.org/Industrial/Industrial_Robot_Driver_Spec

[9] "Add support for async streaming ('joint_command')," December 2017. [Online]. Available: https://github.com/ros-industrial/industrial_core/issues/139

[10] "ABB Externally Guided Motion EGM," February 2017. [Online]. Available: http://new.abb.com/products/ABB.PARTS.SEROP3HAC054376-001

[11] Ø. Mæhre, "Following Moving Objects Using Externally Guided Motion (EGM)." M.S. thesis, University of Stavanger, Norway, 2016.

[12] "Latest robot controller software from ABB combines flexibility and productivity for developers," December 2017. [Online]. Available: http://www.abb.com/cawp/seitp202/2a21e772c71016d1c1257daf004046bc.aspx

[13] "Nitinol Shape Setting," December 2017, Johnson Matthey Medical Components. [Online]. Available: http://jmmedical.com/resources/251/Nitinol-Shape-Setting.html

[14] "34 | OPTIMA Batteries," April 2017. [Online]. Available: https://www.optimabatteries.com/en-us/redtop-starting-battery/34

[15] *Trench T4 Power MOSFET IXTN660N04T4*, IXYS, 2016. [Online]. Available: http://ixapps.ixys.com/DataSheet/DS100728A(IXTN660N04T4).pdf

[16] "Arduino Yún," December 2017. [Online]. Available: https://store.arduino.cc/usa/arduino-yun

[17] *Very Low Noise, 24-Bit Analog-to-Digital Converter*, Texas Instruments, December 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/ads1255.pdf

[18] *Low Power, Single-Supply DIFFERENCE AMPLIFIER*, Texas Instruments, November 1996. [Online]. Available: http://www.ti.com/lit/ds/symlink/ina132.pdf

[19] *INA1x9 High-Side Measurement Current Shunt Monitor*, Texas Instruments, February 2017. [Online]. Available: http://www.ti.com/lit/ds/symlink/ina139.pdf

[20] merose, "AnalogScanner," December 2017. [Online]. Available: https://github.com/merose/AnalogScanner

Appendix A

Hardware Costs

Table A.1: Approximate Component Cost of Robot-Mounted Nitinol Shape Setter

Item	Cost
Battery	$160
Electrical Enclosure with Subpanel	$110
3-stage battery charger	$72
3 Panel-mount LED Indicators	$16
2 5/16" Ceramic Bolts	$14
10 Battery terminals (various sizes)	$30
40 feet 2AWG Welding Cable	$73
Arduino Yún	$68
500 Amp Relay	$50
Fuse and holder	$35
Miscellaneous	?
Total:	About $1000