## Use Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission to download and/or print my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature _____

Date _____

# Development and implementation of a ROS-based reconfigurable controller for a CRS Robot manipulator

by

**Abdulrahman Alshankiti**

A thesis

submitted in partial fulfillment of the requirements

for the degree of

Master of Science in Measurement and Control Engineering

College of Engineering, Idaho State University

Spring 2014

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of Abdulrahman Alshankiti find it satisfactory and recommend that it be accepted.

_____, Major Advisor
Dr. Alba Perez-Garcia (Associate Professor, Associate Chair of Mechanical Engineering)

_____, Co-Advisor
Dr. Marco P. Schoen (Professor of Mechanical Engineering)

_____, Graduate Faculty Representative
Dr. Steve C. Chiu (Associate Professor of Electrical Engineering)

# Acknowledgments

I would like to thank everyone who help me and contributed to my graduate project. I would like to start by thanking my major advisor Dr. Alba Perez-Gracia for all the support and encouragement that made me complete my project. My thanks also to Dr. Marco P. Schoen who helped me choose the right decisions and giving me the opportunity to be one of the Idaho State University graduate students. I also want to thank Dr. Steve C. Chiu for being my Graduate Faculty Representative.

# Contents

# List of Figures

# List of Tables

# Abstract

One of the problems of using robot manipulators in the classroom is that many academic and industrial robots have their own languages, and the learning curve to master those is discouraging for the students. This is especially true with older robots. In this project, an old robot arm CRS-Plus A1500 is to be overhauled so that the students can easily program it using ROS or interfacing with common academic software such as LabVIEW. An NI CompactRIO controller is used because of its reconfigurability. The old robot controllers are discarded and the actuators are connected to new ROB-09670 H-bridge drivers and a 600 Watt power supply, and then to a RIO compact controller. This robot is actuated by five DC field servo motors and has encoders for feedback. Forward and inverse kinematic analysis of the robot are calculated in order for the students to be able to program routines to perform standard tasks. Finally, Labview and ROS are configured to work with each other under a local network to send commands and receive feedback. The system is configured so that ROS will send and receive data packages from the local network and LabVIEW will act at a translator to these packages for ROS. The new system will allow students to experience robot performance as well as learn rudimentary ROS programming.

# Chapter 1

# Introduction

Robots are very important to facilitate teaching and learning process and to accomplish repetitive tasks in industry, see [5], [23], [1]. They are also playing a vital role in research, for testing of new algorithms, and manipulation tasks, see [3]. With the advancement of technology in drivers, control and software, students in school or people in industry can benefit from these technologies to advance their knowledge in the area. Using Internet-based telerobotic systems for remote control, manufacturing, surgery, training, and education has become a big research topic and area [26], [9], [28],[8] .However, older robots with very old controllers and specific programming languages take a longer time to understand and learn the language. This increases the learning curve without adding new features to the robot. Replacing the older controller with a user friendly modular controller with simple and common programming language will enhance the creation of new algorithms. Students can also be familiar with different manipulation tasks and kinematics of the robot easily.

There are variety of programming languages that can be used in the development of the robotics projects such as: Player, YARP[13], Orocos[2], CARMEN[14], Orca[11], MOOS[15], and Microsoft Robotics Studio[10], yes we have a lot of options to choose from. However, they vary from one to another in the perspective of easiness to use and hardware support.

The intuitive way that *LabVIEW* implemented their programming language, made the user concentrate on graphing the layout of his system, and don't worry about writing a code or data structures. Another case of easiness in programming is Robot Operating System (ROS) [22],[25] by making a more liberal approach, by developing set of software libraries and tools that help you build robot applications. Instead of developing a system that serves only one specific robot, ROS is making a universal environment for developing, testing, simulating and controlling multiple robots in respective to the manufacturer, such as [7],[12].

In this research, the controller of an old CRS-Plus A150 robot arm is substituted with a National Instruments CompactRIO Controller, ROS and *LabVIEW* are used as development environments and for cross-platform communication. The final goal is to have the robot arm properly functioning with the new controller. The software and tools used to control the robot are found in chapter two. The architecture of the robot are found in Chapter three. The experiments conducted on the software and hardware are found in chapter four.

We hope that this new setup helps make using the robot useful for educational purposes such as: using a more popular programming languages, and understanding how controllers work.

# Chapter 2

# Software setup

This chapter introduces the software needed to control the robotic arm using National Instruments hardware, *LabVIEW* and Robot Operating System. There is no official driver to control the compact rio driver directly using linux, ROS and *LabVIEW*. Here, the brief history and use of each softwares are discussed. The connection between the two different operating systems to gain access to the controller is established.

## 2.1 Robotics Operating System (ROS)

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications, from drivers to state-of-the-art algorithms, and with powerful developer tools[27]. This system can operate like any operating system and provide a lot of functionality like, hardware abstraction, low level device control, massage parsing and package management. It also works like a programming language because it has tools and libraries that can help in building, writing, and running code across different platforms. In other words, ROS frameworks can be compared to Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio [27].

One great example of the power of ROS is the real-time view of the robot or

graph mode that is a peer to peer communications coupled loosely using one of the many communication infrastructure like synchronous communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server [25].

### 2.1.1 File-systems

ROS mainly covers six types of file-system structures [25].

- Packages: a package could contain dataset, library,configuration file, runtime processes(nodes) or anything organized together. This it the main data type that will organize developed software in ROS.
- Meta-packages: related group of packages that serve a certain case. In other words, to solve compatibility issues.
- Package Manifests: an XML file to show how should the package file be constructed and will include description, name, version, license, dependences and other important information.
- Repositories: common version control system witch share a collection of packages.
- Message: ROS ueses a certain data structure to send messages.
- Service: ROS ueses a certain data structure to send request and receive them.

### 2.1.2 ROS Computation Graph Level

Processing data together in a peer to peer network manner in ROS is called the computation graph. Concepts of the computing graph in ROS are as follow [25]:

- Nodes: Divide the control processes of a robot into nodes each node will be responsible for one single task. This serve the main reason that the programming in ROS is modular.

- Master: To exchange messages or start a service, nodes should be able to find each other. Here the master will help to provide registration and lookup.

- Parameter Server: a central location that allow data storage with a master key. For now its part the master services.

- Messages: Passing messages between nodes to communicate with each other. The structure of the message consists of standard data types, as well as arrays and nested structure.

- Services: The communication part is done by the the publish and subscribe model and is very flexible. When a node request a service it will define two messages one for the request and one for the reply, as shown in Figure 2.1[25]

- Topics: The content of a message is given a name that is a topic to identify it. The topic acts like a route for the transportation system with publish and subscribe semantics between nodes. When a node is looking for a specific data it will subscribe to the topic with the needed information. Furthermore, multiple subscription and publication can be established by each note in a system.

- Bags: This concept act as a back-up system for the message data.

For the purpose of this project, Topics are the most important element here, because it carries the data to and from nodes. we are trying to intercept this flow of data here is where ROS toolkit and Rosbridge come to action, by making the flow of data understandable, and easer to manage.

## 2.2 *LabVIEW* System Design Software

*LabVIEW* can create a variety of applications that can relate to real-world situations in fields like science and engineering, with a more efficient development environment. In *LabVIEW* we have the resources and options to do different types of test, measurements, and control applications [20].

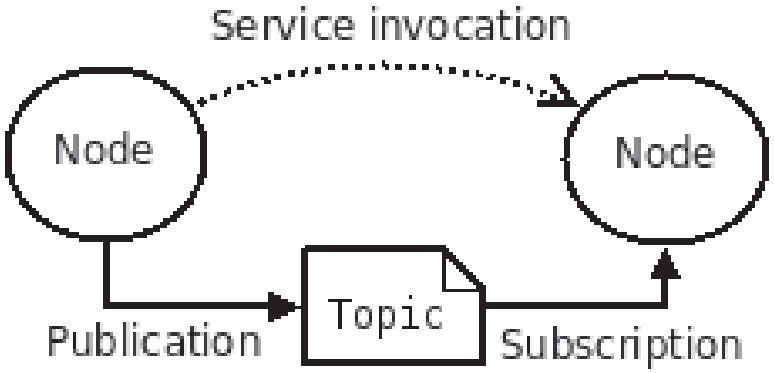



Figure 2.1: Service invocation [25]

### 2.2.1 G programming language

G programming language is graphical data-flow charts that can tie together data acquisition, analysis, and logical operation. the processing of the data will be in real-time as they come available [20]. It is more practical than the sequential programming languages, we don't have to worry about memory allocation and language syntax. we only have to lay out our flow diagram and the wires between the(connection between each node) [20].

### 2.2.2 Hardware support

In this project national instrument controller is used. *LabVIEW* can support lots of hardware such as [20]:

- Scientific instruments.
- Data acquisition devices.
- Sensors.
- Cameras.
- Motors and actuators.

The NI CompactRIO controller is used to control the robot arm in this project, chapter three will go in detail describing the controller.

### 2.2.3 *LabVIEW* Example of ROS toolkit

The installation of ROS toolkit adds a new set of blocks to the *LabVIEW* wide library of functions, to allow the ability to communicate with the ROS computer. three of those new blocks are used in the Figure 2.2. Each one is clearly label and has its own description next to it. The following point explains each one of them [4].

1. The open connection block: to establish a connection with the ROS computer, by providing the IP address of it, at the same time have the Rosbrige package running in the ROS computer.

2. The send data block: this block make to data calculated in *LabVIEW* understandable to ROS by using the publish and subscribe semantics.

3. Close connection block: to terminate the connection between ROS and *LabVIEW*.

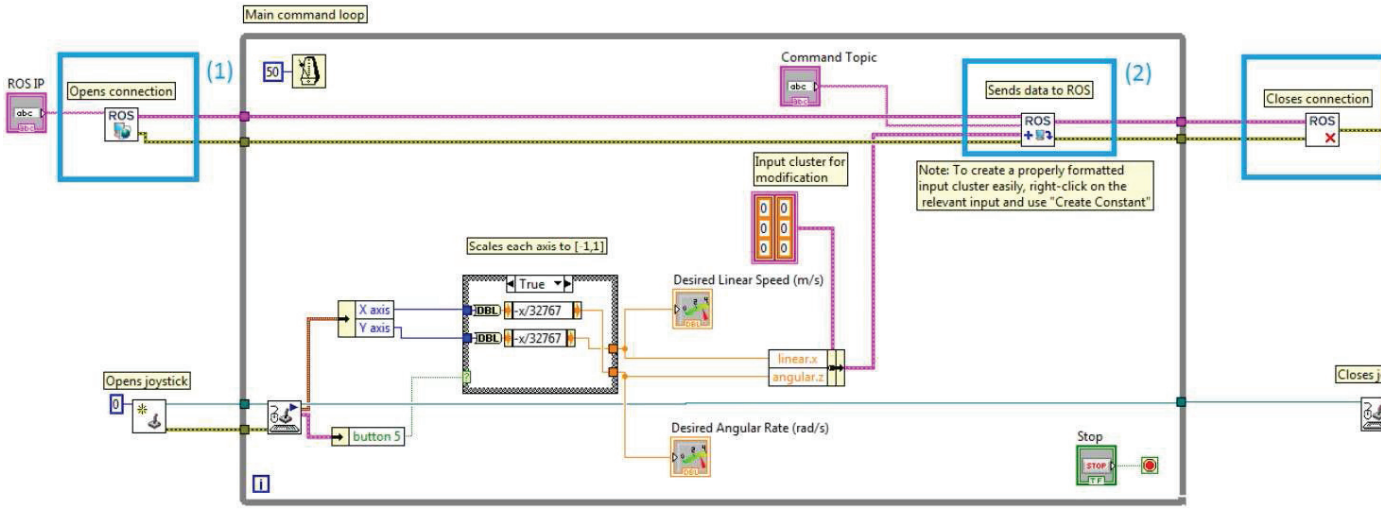



Figure 2.2: Joystick Demo [4].

## 2.3 Connectivity setup

To start working on ROS and *LabVIEW* the following steps are followed to ensure the proper connection between all the devices that are used. The devices used in this project are:

- A dedicated computer for Linux Operating System.
- A dedicated computer for Windows Operating System.
- A router or a switch.
- CompactRIO controller.

### 2.3.1 Local network

There is a need for a local network to make all the hardware communicate with each other, a computer running Windows Operating System, another computer running Linux and the CompactRIO controller. All of them will be on the same local network with static IP. it simply can be done by a switch or router. to obtain the IP address for the controller, it is possible to access it from the router web UI or using the measurement and automation explorer program as show in the Figure 2.4

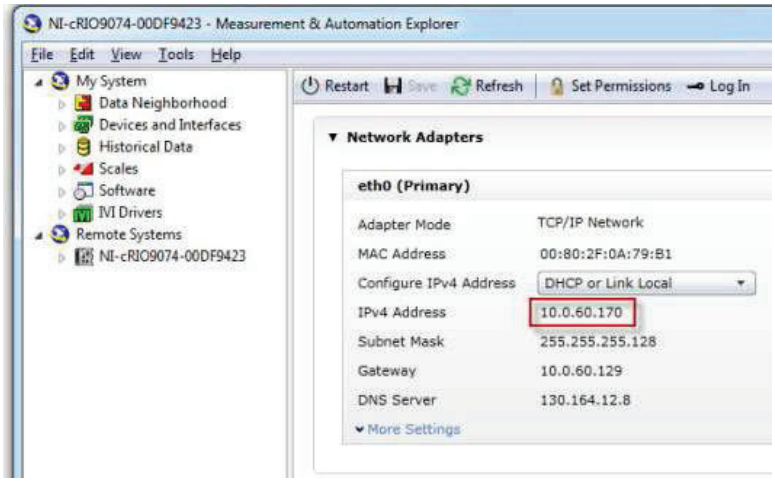



Figure 2.3: IP address for the NI Compact RIO controller in MAE


Figure 2.4: The router

### 2.3.2 Ubuntu machine

Currently ROS only runs on Linus or Unix based devices. It is recommended to use currently Ubuntu 12.04 LTS due to its stability. the next points will have some sudo command that needs to run in the terminal to have the ROS Fuerte distribution installed [24]:

- sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" ¿ /etc/apt/sources.list.d/ latest.list'
- wget http://packages.ros.org/ros.key -O - — sudo apt-key add -
- sudo apt-get update
- sudo apt-get install ros-fuerte-desktop-full
- echo "source /opt/ros/fuerte/setup.bash" ¿¿ /.bashrc. /.bashrc
- sudo apt-get install python-rosinstall python-rosdep

After installing ROS, there is one more step that is installing the Rosbridge package, and to do that navigate to the semantic package manager. next, in the search bar

write Rosbridge, and install the package that compatible with the fuerte distribution of ROS.

To realize the importance of the Rosbridge package, Figure 2.5 illustrates the need to use this package.

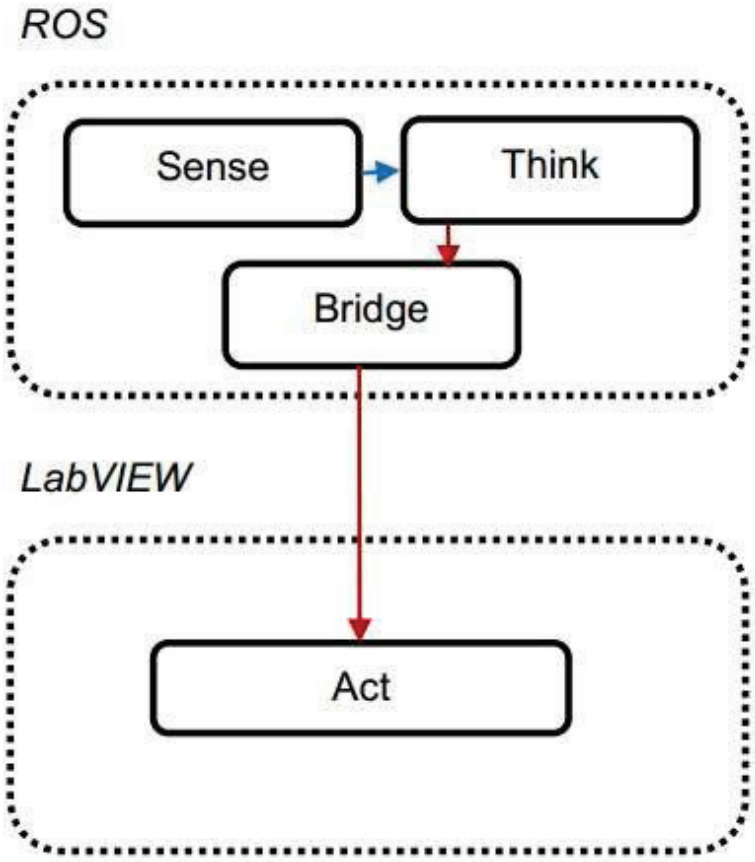



Figure 2.5: *LabVIEW* controlled robot

### 2.3.3 Windows machine

Currently *LabVIEW* runs on windows based machines, in this project, *LabVIEW* 2012 with the following package is installed on the computer:

- FPGA module: Field-Programmable-Gate-Array, is a module to enable the process of controlling or measuring any I/O signals, without the need to the low level programming languages[18].

- Real-Time module: Real-Time is an extension to *LabVIEW* development system to allow over the network communication to run a project on an embedded systems such as NI CompactRIO[19].

- ROS toolkit: ROS toolkit in an add-on package that allows *LabVIEW* communicate with ROS. Note that it is necessary to install the Rosbridge package establish connection[4].

### 2.3.4 Controller

The drivers and software are installed in the windows machine before the controller is used. This step will be performed in the windows computer. The following steps will demonstrate the procedures to have the driver up and running. First, install all *LabVIEW* software required to run by the driver, as follow:

- NI-*LabVIEW* 2012 or later version.

- NI-RIO Driver.

- NI-FPGA Module.

- NI-Realtime module.

After getting done with installing all required software, there is the need to do the same steps to the controller itself, because the controller has an internal storage that should contain all softwares and drivers needed to have the controller function by its own through the network. These software include modules information libraries and interfacing software.
Using the Measurement and Automation Explorer, with proper connection to the local network navigate to the software icon to start configuring the software needed to start using the controller as shown in Figure 2.6.

Next step, install all the software that have the check mark as shown in the Figure 2.7.

Finally, the controller will reboot and will be ready to use.

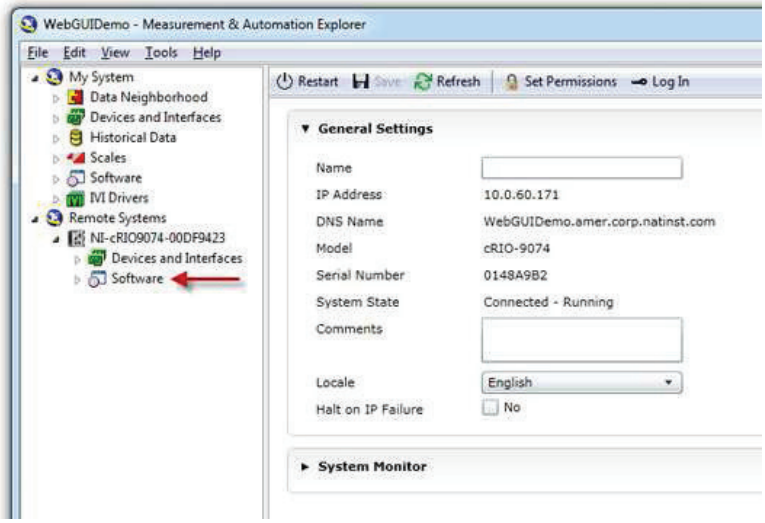



Figure 2.6: Configuring the controller

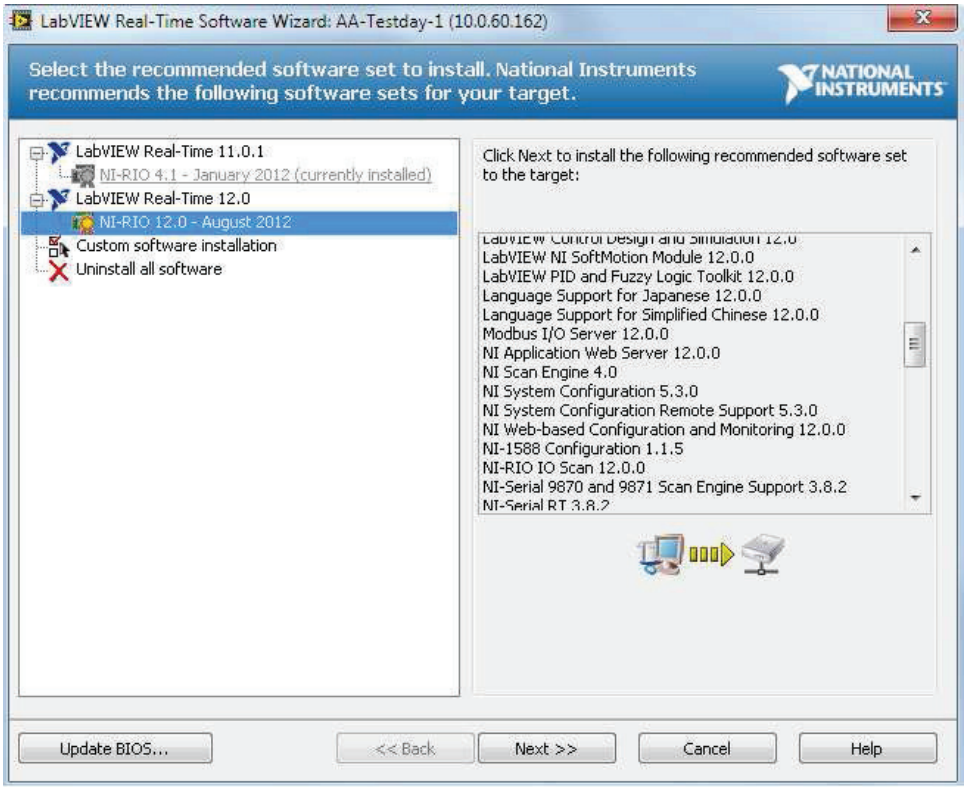



Figure 2.7: All needed software to be installed

## 2.4 Visual model of the robot in ROS

### 2.4.1 The simulation structure

The robot unified description format is a package in ROS that has a C++ parser and a number of XML specification for sensors, scenes and robot models. URDF package consists of a number of different packages some of them are ROS packages to parse and generate the model, and some of them are debian packages for data structure and interactive 3D application.

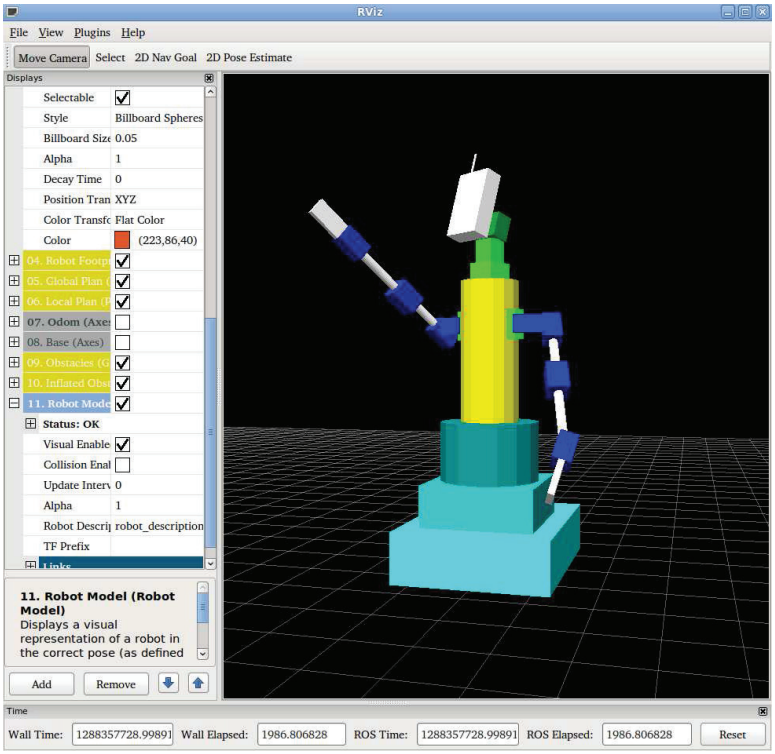



Figure 2.8: Example of the PI-robot in URDF

### 2.4.2 Simulated CRS-Plus robot arm

The simulation is constructed in the URDF package, using the XML model specification library its possible to reach a design similar to the CRS-Plus robot arm, as in Figure 2.9. The code for the URDF design is found in the appendix.

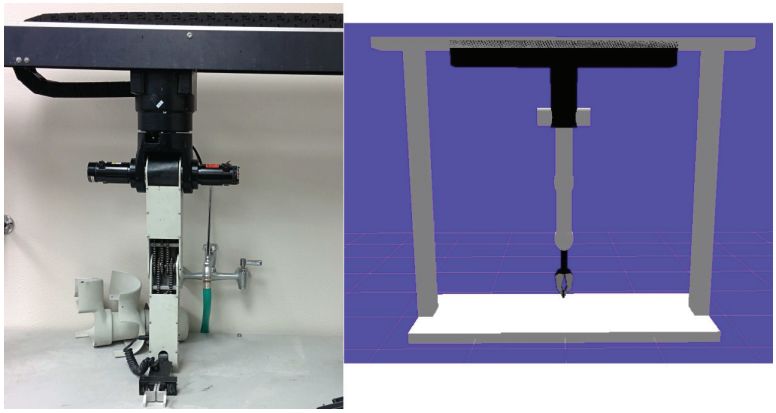


Figure 2.9: CRS-Plus robot arm simulation

The robot kinematics is stored in a tree structure as shown in Figure 2.10, where the joints, first translation, and then five revolute joints, dubbed the waist, shoulder, elbow, wrist roll, and wrist pitch. Each joint has a XML model specification.

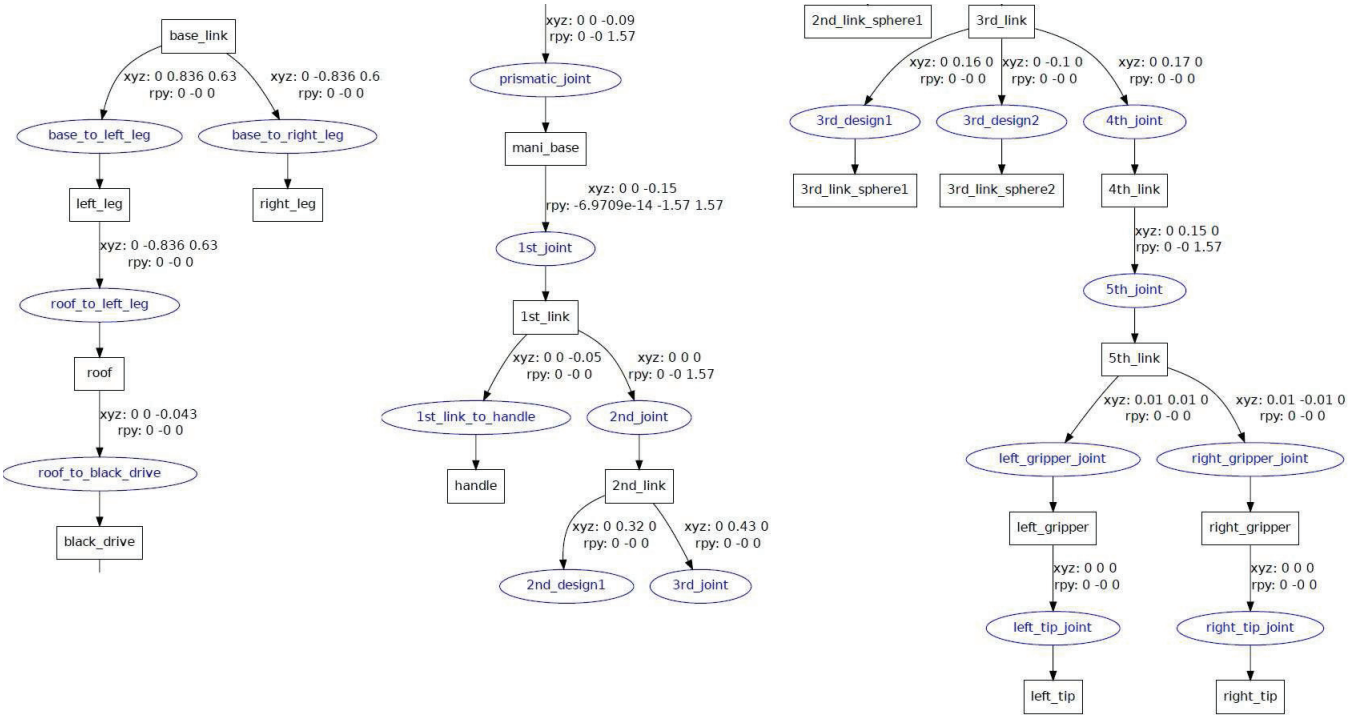



Figure 2.10: Tree structure

The joint state publisher dialog box as shown in Figure 2.11 can manipulate the angle of each joint, and give every possible degree of freedom. Figure 2.12 and 2.13

show the axis for each joint with respect to the local coordinate frame.

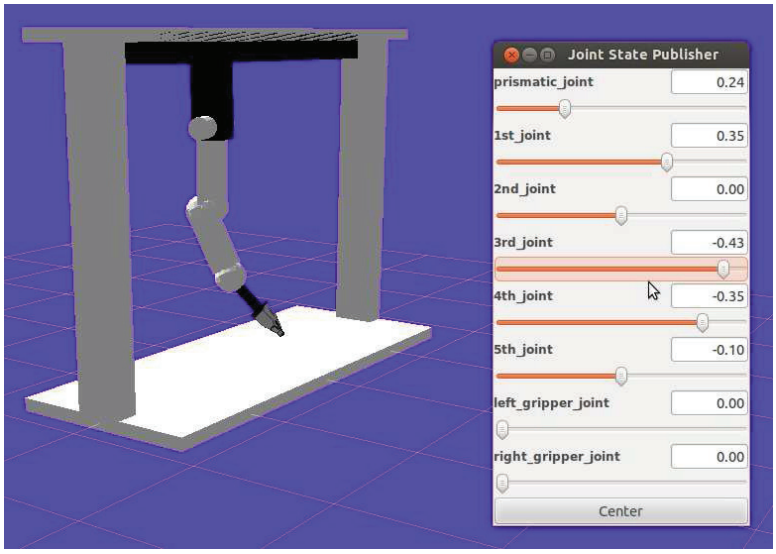



Figure 2.11: Joint state publisher

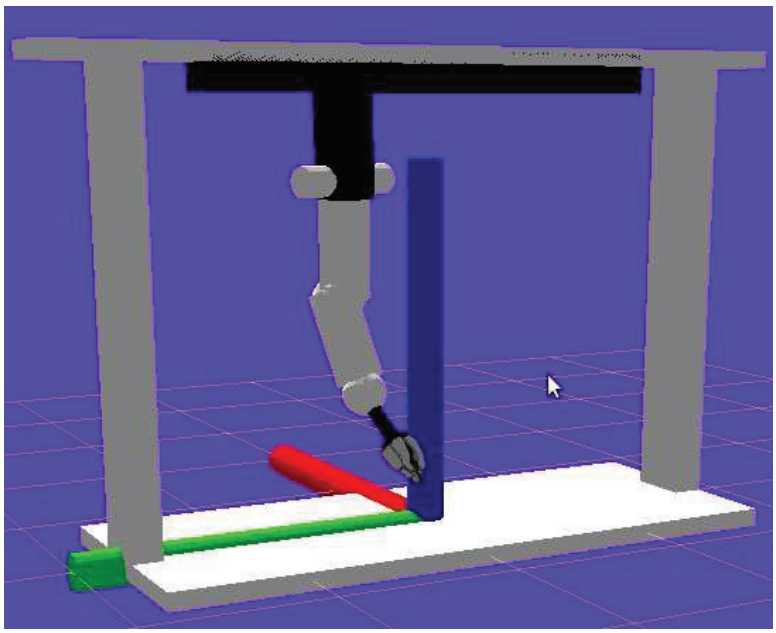


Figure 2.12: Base axis

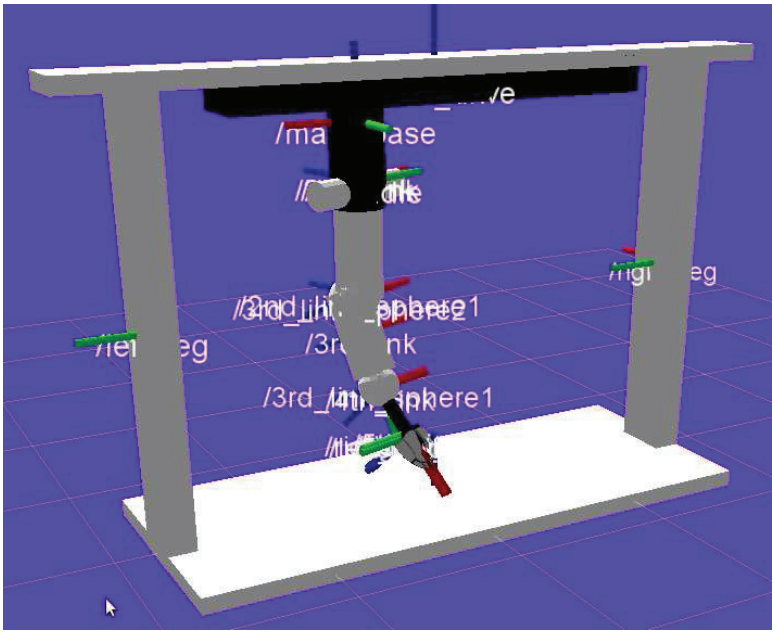


Figure 2.13: Joints' axis

### 2.4.3 Summary

In summary, in this Chapter the different pieces of software used for integrating robot simulation, path planning, generation of control commands and interfacing are presented. All these are either open-source or widely used in academia, so that the learning curve for students is expected to be small.

# Chapter 3

# Hardware

In this Chapter, the different pieces of hardware needed for having a fully operative robot are presented and described in detail. These are listed below:

- CRS-PLUS A150 robot arm and controller.
- NI CompactRIO controller and its modules.
- Power amplifier and driver for driving the motors.
- Power supply.

Figure 3.1 shows how these hardware are connected to each other and used in this project.

## 3.1 CRS-PLUS A150 robot arm

This robot is one of the A100/200 series of small industrial robot system, and come with the RSC-P8 controller and configured to run the ROBCOMM communication package. It used to be a popular academic robot, which is no longer manufactured. Due to this reason, it is difficult to find instruction manuals, repair parts and information on the programming language.

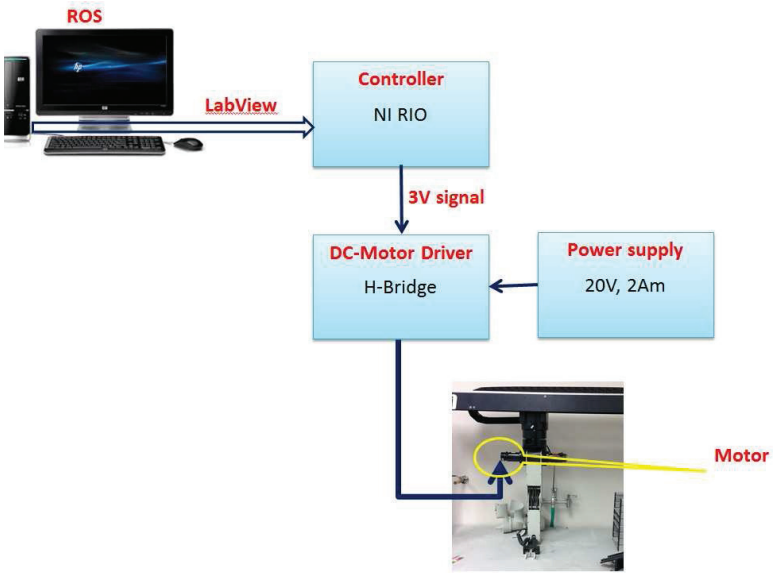



Figure 3.1: Overview of the hardware

### 3.1.1 Robot arm

The setup of this CRS robot arm existing at the ISU Robotics laboratory consists of six degrees of freedom robot arm, a first translation, and then five revolute joints, dubbed the waist, shoulder, elbow, wrist roll, and wrist pitch. Table 3.1 shows the range of motion for each joint, except the first prismatic joint, which has an approximate range of approximately 24 inches. Figure 3.3. shows a schematic of the robot joints.

### 3.1.2 RSC-P8 controller

This robot system is controlled by the RSC-P8 controller in Figure 3.4 that has proportional P-type axis cards, shown in Figure 3.5. The controller took the feedback signal from the encoder to create a closed-loop controlling scheme. The card takes a signal from the motherboard of the controller, change it to an analog signal, so it can be sent to amplifier to drive the motors. The card also read the feedback that has been sent from the incremental optical encoder (shown in Figure 3.6) in a square wave form (channel A, channel B, and the zero crossing index signal)

Table 3.1: Robot joints

| Joint | Range of motion in Deg |
|---|---|
| Base | - or + 175 |
| Shoulder | 47 -0 or +100 |
| Elbow | -130 or +0 |
| Wrist pitch | - or + 110 |
| Wrist roll | - or + 180 |

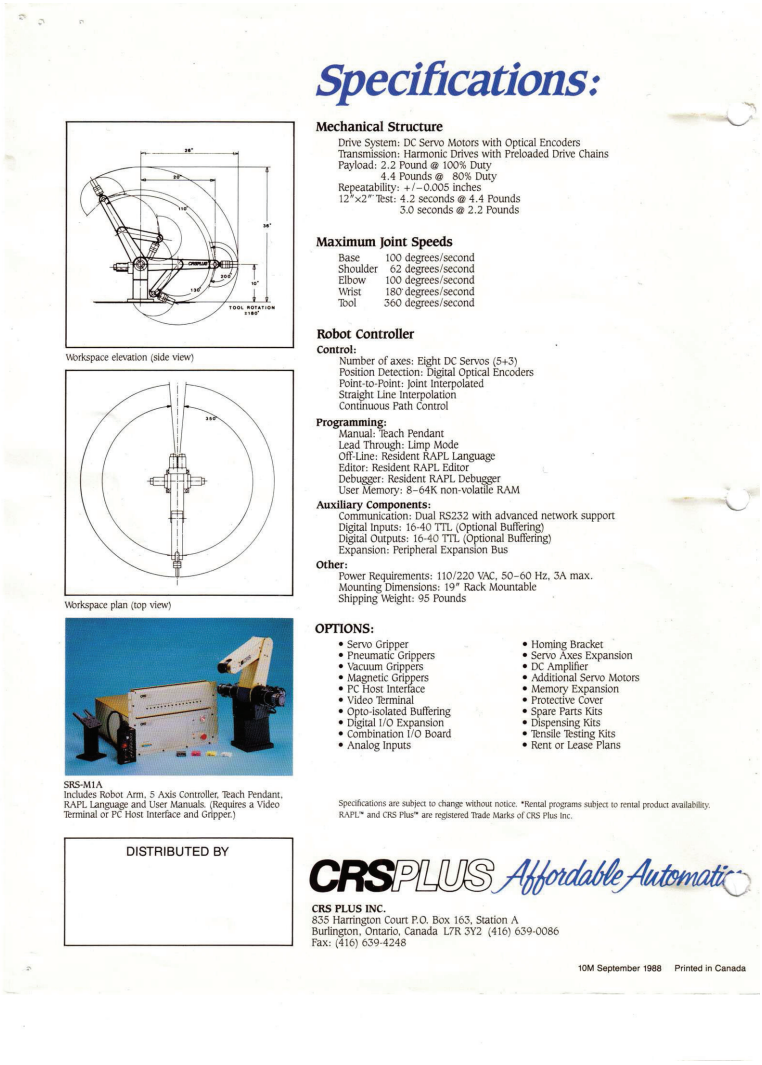
### 3.1.3 Joint's motor

This robot arm uses an industrial grade motors that are permanent magnet DC motors and also includes an incremental optical shaft position encoder as in Figure 3.6. the optical encoder operates as follow:

- Two disks with fine grating, one is fixed, the other is rotating with the the motor.
- With the rotation of the motor shaft light goes through the fine grating of the two disks.
- Then there is a photo-cell that can catch this light and then convert it to voltage
- The voltage pulses can be timed and counted to compute the relative position and velocity of the shaft.

### 3.1.4 Specification for joints' speed

The table 3.2[21] show the speed of five motions the robot arm can achieve, and this play a huge roll in finding the velocity of each joint.



# Specifications:



Workspace elevation (side view)



Workspace plan (top view)



SRS-M1A
Includes Robot Arm, 5 Axis Controller, Teach Pendant, RAPL Language and User Manuals. (Requires a Video Terminal or PC Host Interface and Gripper.)

DISTRIBUTED BY

**Mechanical Structure**

Drive System: DC Servo Motors with Optical Encoders
Transmission: Harmonic Drives with Preloaded Drive Chains
Payload: 2.2 Pound @ 100% Duty
4.4 Pounds @ 80% Duty
Repeatability: +/– 0.005 inches
12"×2" Test: 4.2 seconds @ 4.4 Pounds
3.0 seconds @ 2.2 Pounds

**Maximum Joint Speeds**

| | |
|---|---|
| Base | 100 degrees/second |
| Shoulder | 62 degrees/second |
| Elbow | 100 degrees/second |
| Wrist | 180 degrees/second |
| Tool | 360 degrees/second |

**Robot Controller**

**Control:**
Number of axes: Eight DC Servos (5+3)
Position Detection: Digital Optical Encoders
Point-to-Point: Joint Interpolated
Straight Line Interpolation
Continuous Path Control

**Programming:**
Manual: Teach Pendant
Lead Through: Limp Mode
Off-Line: Resident RAPL Language
Editor: Resident RAPL Editor
Debugger: Resident RAPL Debugger
User Memory: 8–64K non-volatile RAM

**Auxiliary Components:**
Communication: Dual RS232 with advanced network support
Digital Inputs: 16-40 TTL (Optional Buffering)
Digital Outputs: 16-40 TTL (Optional Buffering)
Expansion: Peripheral Expansion Bus

**Other:**
Power Requirements: 110/220 VAC, 50–60 Hz, 3A max.
Mounting Dimensions: 19" Rack Mountable
Shipping Weight: 95 Pounds

**OPTIONS:**

- Servo Gripper
- Pneumatic Grippers
- Vacuum Grippers
- Magnetic Grippers
- PC Host Interface
- Video Terminal
- Opto-isolated Buffering
- Digital I/O Expansion
- Combination I/O Board
- Analog Inputs
- Homing Bracket
- Servo Axes Expansion
- DC Amplifier
- Additional Servo Motors
- Memory Expansion
- Protective Cover
- Spare Parts Kits
- Dispensing Kits
- Tensile Testing Kits
- Rent or Lease Plans

Specifications are subject to change without notice. *Rental programs subject to rental product availability.
RAPL™ and CRS Plus™ are registered Trade Marks of CRS Plus Inc.

CRSPLUS Affordable Automati...

**CRS PLUS INC.**
835 Harrington Court P.O. Box 163, Station A
Burlington, Ontario, Canada L7R 3Y2 (416) 639-0086
Fax: (416) 639-4248

10M September 1988 Printed in Canada

Figure 3.2: Robot specifications

### 3.1.5 ROBCOMM-II development software

ROBCOMM-II is a development environment that uses RAPL-2 programming language, this system is exclusive for the CRS-Robotics Company. Writing a code in RAPL-2 is not that easy because it has a long list of subprograms that are not familiar to most of the new users of the language. Figure 3.7[6] shows a general example the the RAPL-2 programming language.

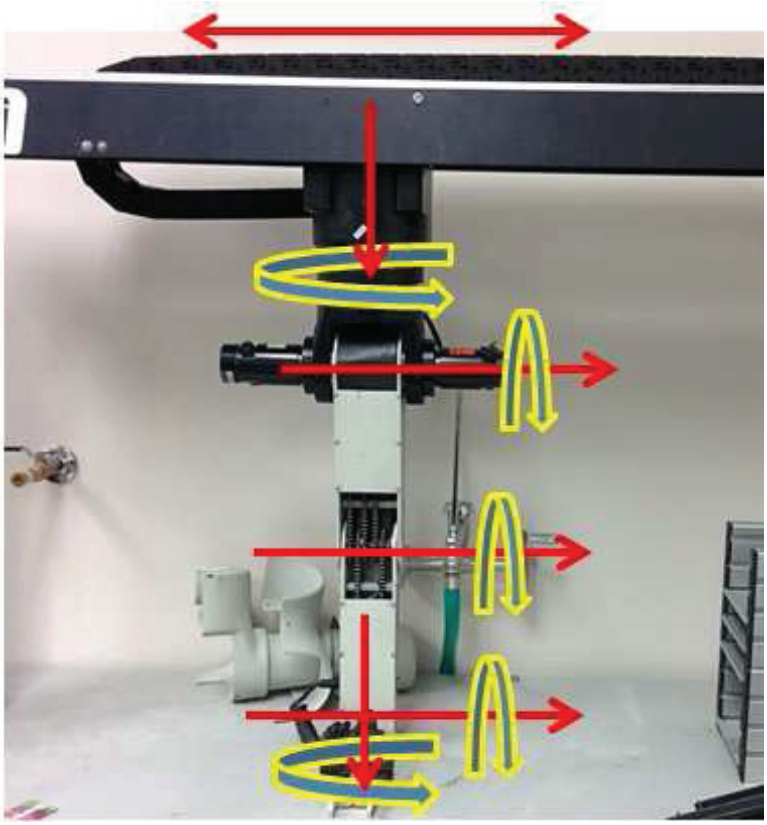


Figure 3.3: CRS-PLUS robot arm

## 3.2 NI CompactRIO controller and modules

NI compactRIO has reconfigurable chassis as shown is Figure 3.8that can host multiple input and output modules and controller embedded inside of chassis for real-time processing and field programming gate array tasks. the communication to this controller can be established by two 100 MB/s ethernet ports, which provide with the ability to program it over the network[17].

### 3.2.1 Modules

The configuration of the CompactRIO controller that we have contains six input and output modules in Figure 3.9, each module has its specific purpose.

The first module (NI 9425) from the left is for sinking digital input signal, it can support up to 32 channels, and can handle voltages up to 24 V. the second modules

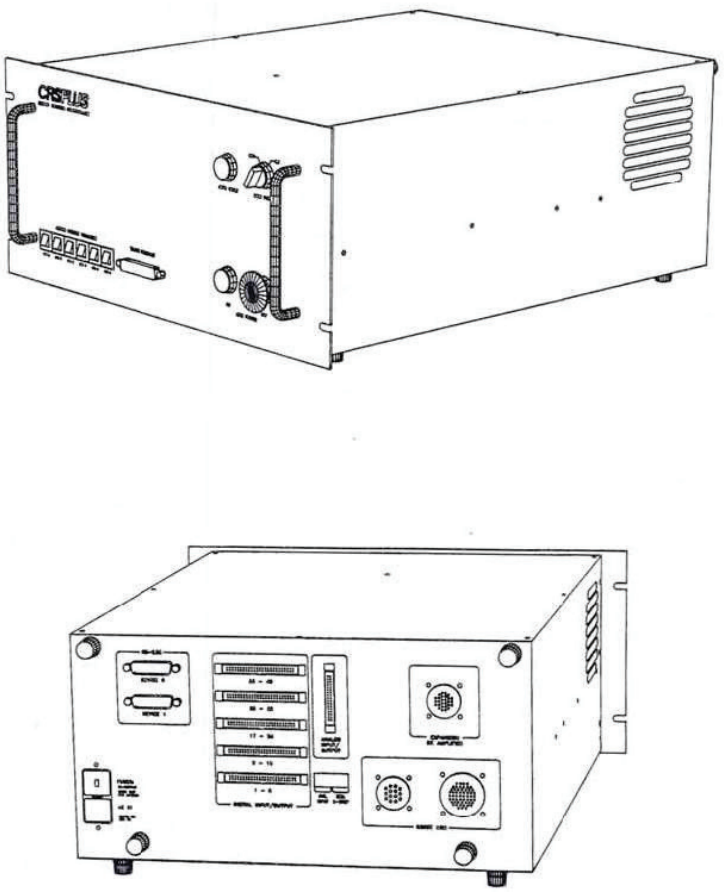


Figure 3.4: Robot system controller

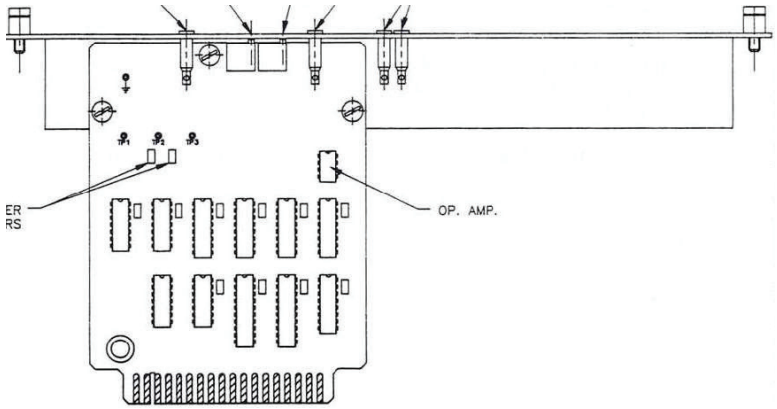



Figure 3.5: P-type servo axis card

(NI 9476) is for souring digital output signal, can support up to 32 channels, and with output range of 6 to 36 V.the third and fourth modules (NI 9222-9223) are for

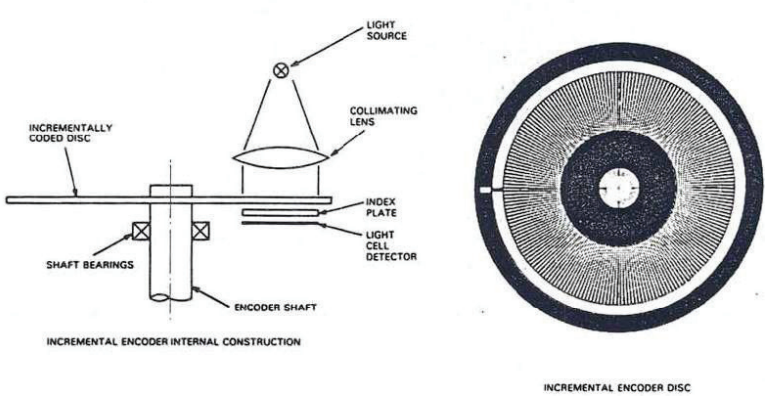



Figure 3.6: Optical encoder layout

Table 3.2: Joints' speed

| Motion | Maximum speed |
|---|---|
| Base | 100Deg/Sec |
| Shoulder | 62 deg/Sec |
| Elbow | 100 Deg/Sec |
| Wrist | 180Deg/Sec |
| Tool | 360Deg/Sec |

analog input signal, it can handle up to +/- 10 VDC. the fifth and sixth modules (NI 9263) are for analog output signals, and can handle up to +/- 10 VDC[16].

## 3.3 Power amplifier and driver for driving the motors

The power supply shown in Figure 3.10 is chosen because of its ability to output up to 22.3 voltages and up to 21.2 AMPs. This is enough to power all motor in the robot arm at the same time if needed.

```
main                ;; begin program

  fast = 50         ;; implicitly declare and initialize integers
  slow = 25
  z = 1

  speed(fast)       ;; set speed
  move(_safe)       ;; move and implicitly declare cartesian location

  do                ;; begin do loop

    appro(_a,5)     ;; pick from location a, implicitly declare location
    grip_open(100)
    grip_finish()
    move(_a)
    finish()
    grip_close(100)
    grip_finish()
    depart(5)

    move(_safe)     ;; move to safe location between pick and place

    appro(_b,5)     ;; place at location b, implicitly declare location
    move(_b)
    finish()
    grip_open(100)
    grip_finish()
    depart(5)

    move(_safe)     ;; move to safe location between place and pick
    z = z + 1       ;; increment counter in loop

  until z == 10     ;; condition to end do loop

end main            ;; end program
```

Figure 3.7: Simple example of RAPL-2 code

Figure 3.8: CompactRIO chassis and controller

The input signal coming from the CompactRIO controller is not sufficient to power the motors due to the limitation of the analog input module. So, the signal is amplified by using ROB-09670 H-bridge as shown in Figure 4.4, this driver can handle motors with separate signals at the same time. It also features a motor supply up to 35


Figure 3.9: CompactRIO controller with modules installed

voltages and up to 2AMPs output power for each. the 10 pins connector beside the green power supply input connector is used for engaging the driver to send power to the motors.


Figure 3.10: 600W Power supply

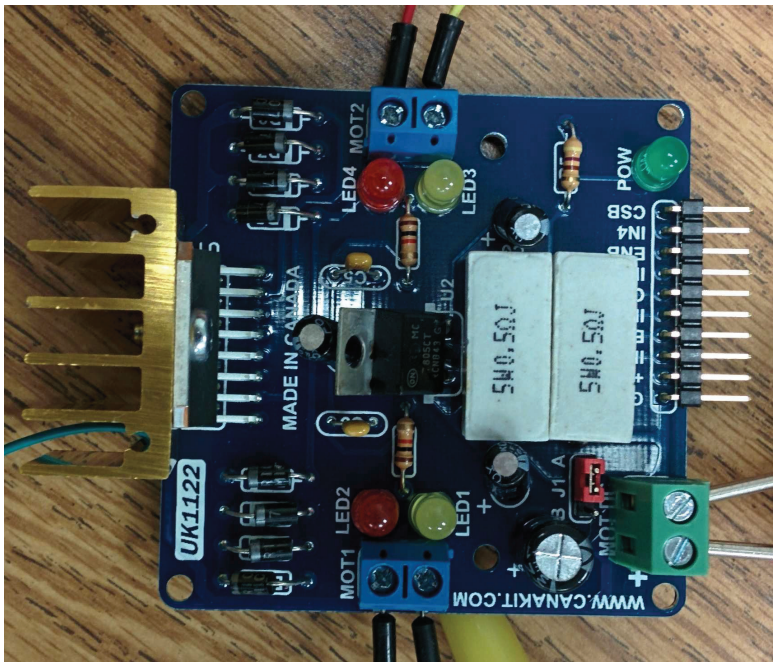



Figure 3.11: ROB-09670 Motor driver

## 3.4 Summary

In this Chapter, the hardware associated with the project has been presented. This includes the mechanical structure of the robot, the old controller and the new controller, driver and power supply systems that are going to be used instead of it. This has the advantage of freeing the robot arm, so it can be used with different controller not just the CompactRIO controller.

# Chapter 4

# Experiments and Results

This Chapter discusses the challenges faced with porting the robot arm to the new controller, *LabVIEW* and ROS to make it function with the robot arm properly.

## 4.1 Disassembling the old controller

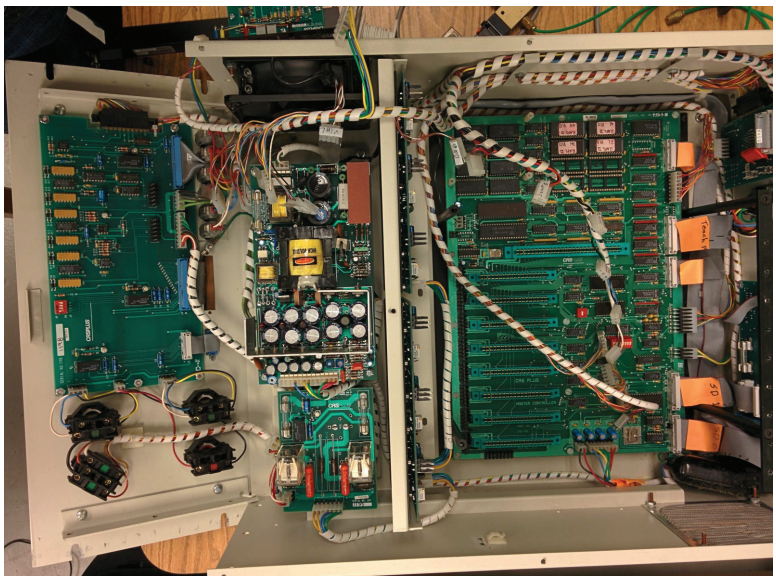


Figure 4.1: The old controller

The Figure 4.1 shows the old controller, on the left hand side there are the power supply and the power amplifiers for each motor. on the right had side there is the motherboard that handles all the controlling tasks. after consulting the robot manual and testing all the wiring harnesses, it was easier to isolate all important wires to make the robot arm work with the CompactRIO controller.


Figure 4.2: Feedback connector for one of the motor

The Figure 4.2 shows one of the six feedback signal wires that are coming from the optical incremental encoder. the first top wire is for providing power to the encoder and it needs 5 VDC to make it work properly.the bottom three are for the feedback signal and they correspond to channel B, channel A and the zero index.

next, powering the motors is done by supplying it with 20 VDC with 2 AMP current for each motor. To do that, use one of the six cables that are labeled joint from 1 to 6. The motor can move in two directions through changing the voltage positive to negative.

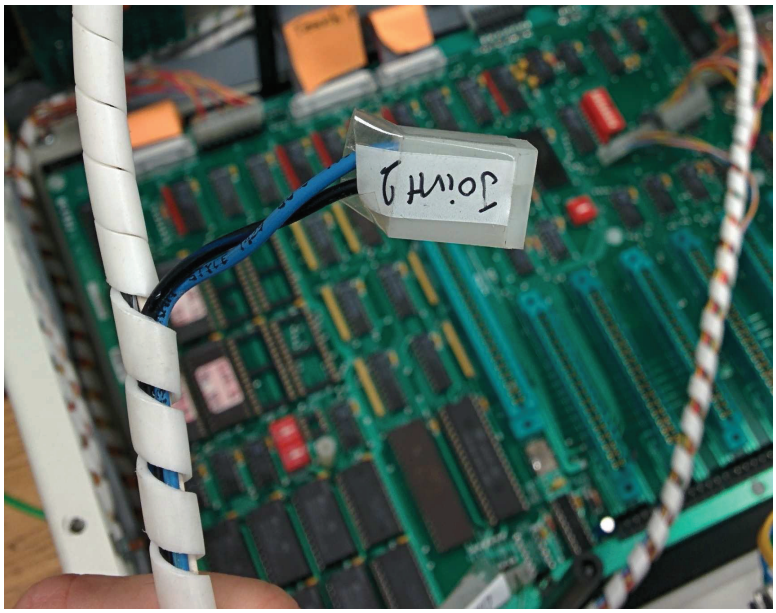



Figure 4.3: Motor connector

## 4.2 Driving the motors

from the specification of the analog input module we see that it does not provide enough voltage and current, so a driver is needed to make the DC motors function properly. after selecting multiple options and testing all of them, the driver that would serve our needs will be the ROB-09670 H-bridge. The signal sent from the controller through one of the four channels in the analog input module, then passed to in input1 and input 2 pins in the driver.

The driver supports up to 2 motors, they can be connected to the blue connectors on each side of the driver. The power supply connects to the green connector with polarity aligned. Figure 4.4 shows a closer look of the driver.

## 4.3 Testing the motors with batteries

Powering the motor with batteries is used here to isolate which motor creates the motion in each of the six joints. After that, each motor connector is labeled with a number that matches the corresponding joint.

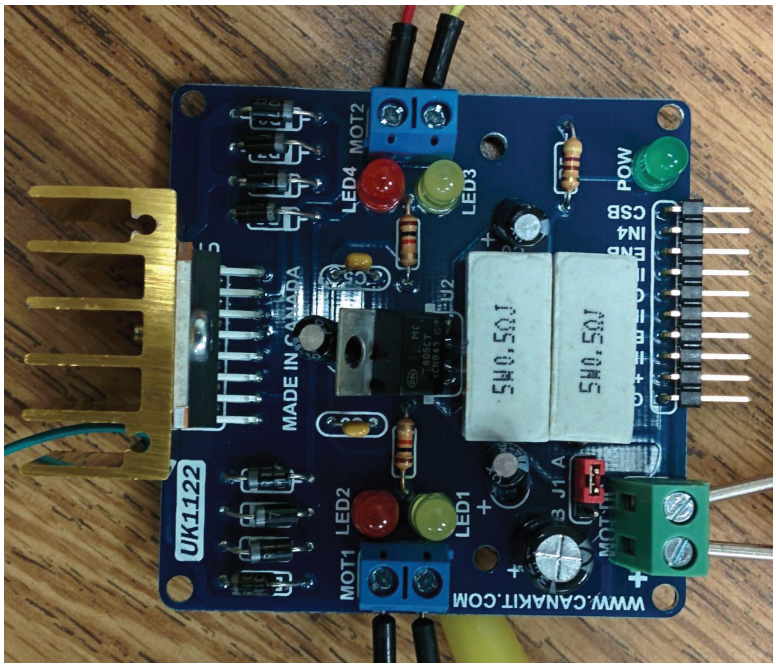



Figure 4.4: The driver

## 4.4 Extracting encoder signal

this test consumed a lot of time to find the right way to read the feedback signal, because the encoder that is inside the motor was power by the motherboard of the old controller. Then the solution is to supply power to the encoder using the V pin in the feedback wire connector. Measuring the signal created by feeding one of the motors with power or by moving it by hand with the National Instrument ELVIS board read the signal produced by the movement of a joint. this procedure has been repeated multiple times, till all feedback wires been assigned to the correct joint.

## 4.5 Testing ROS-*LabVIEW*

This test is conducted by running a Joystick demo controller from the Windows computer throgh LavVIEW and a real-time simulation of the turtlebot robot in ROS computer. First , in ROS computer three important packages are installed through the semantic package manager as follows [4]:

- Remotelab: this package used to drive the robot by the Joystick.
- Turtlebot-simulator: A Real-time simulation of a robot as shown in Figure 4.5[4].
- Rosbridge: this Package is used to allow *LabVIEW* and ROS exchange messages.

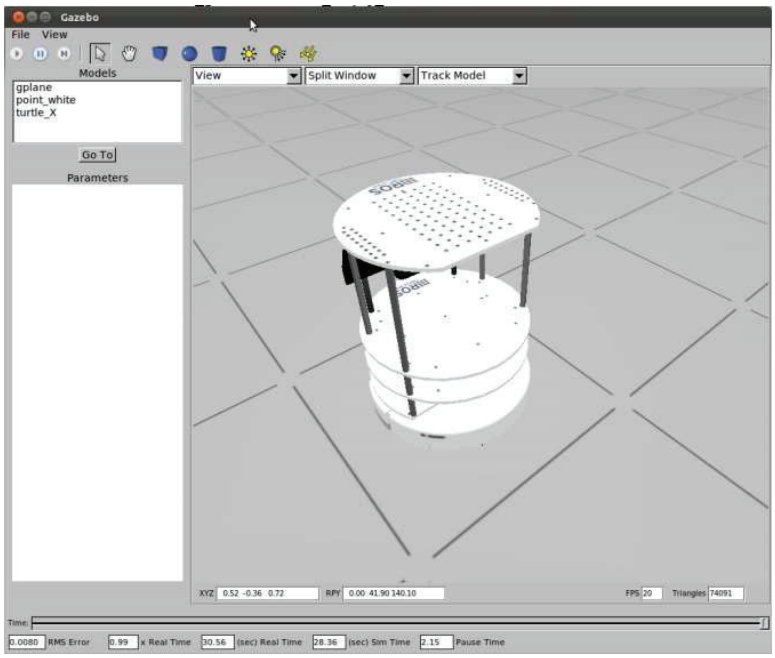



Figure 4.5: TurtleBot Real-Time simulation

Then the simulation is initiated and Rosbridge package by the following sudo commands:

- roslaunch turtlebot gazebo turtlebot empty world.launch
- rosrun rosbridge rosbridge.py

On the Windows computer a Joystick example is executed, to establish a connection with the ROS computer as shown in Figure 4.6[4]. When an input is given by the connected Joystick the Robot moved.

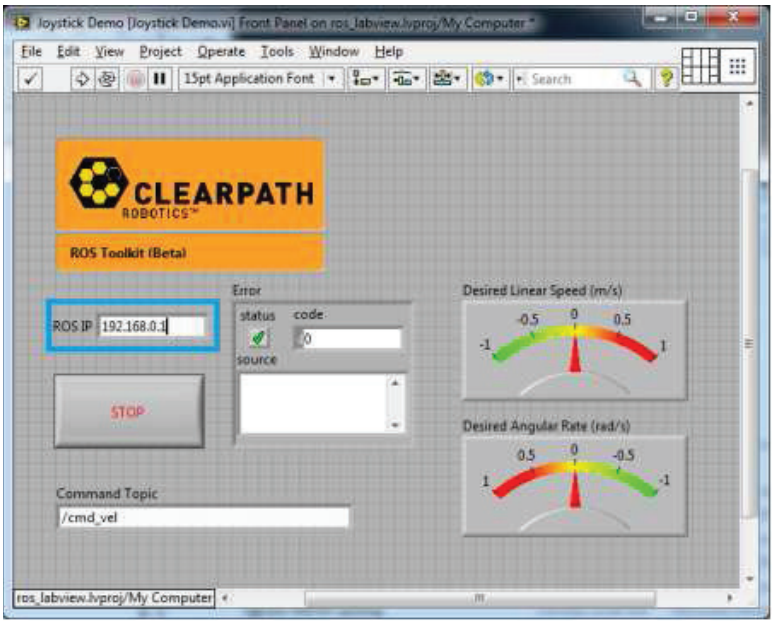



Figure 4.6: Example of *LabVIEW*-ROS toolkit

## 4.6 Feedback signal simulation

Knowing the location of a joint is one of the advantages of ROS, that would help us obtain the angle value of each joint that is needed to reach a final position, that is, giving initial and final positions to each angle and computing a trajectory from initial to final position. The following Figure 4.7 shows the change of angle in respect to time, as time increases the angle changes till the robot arm reach the optimal position.

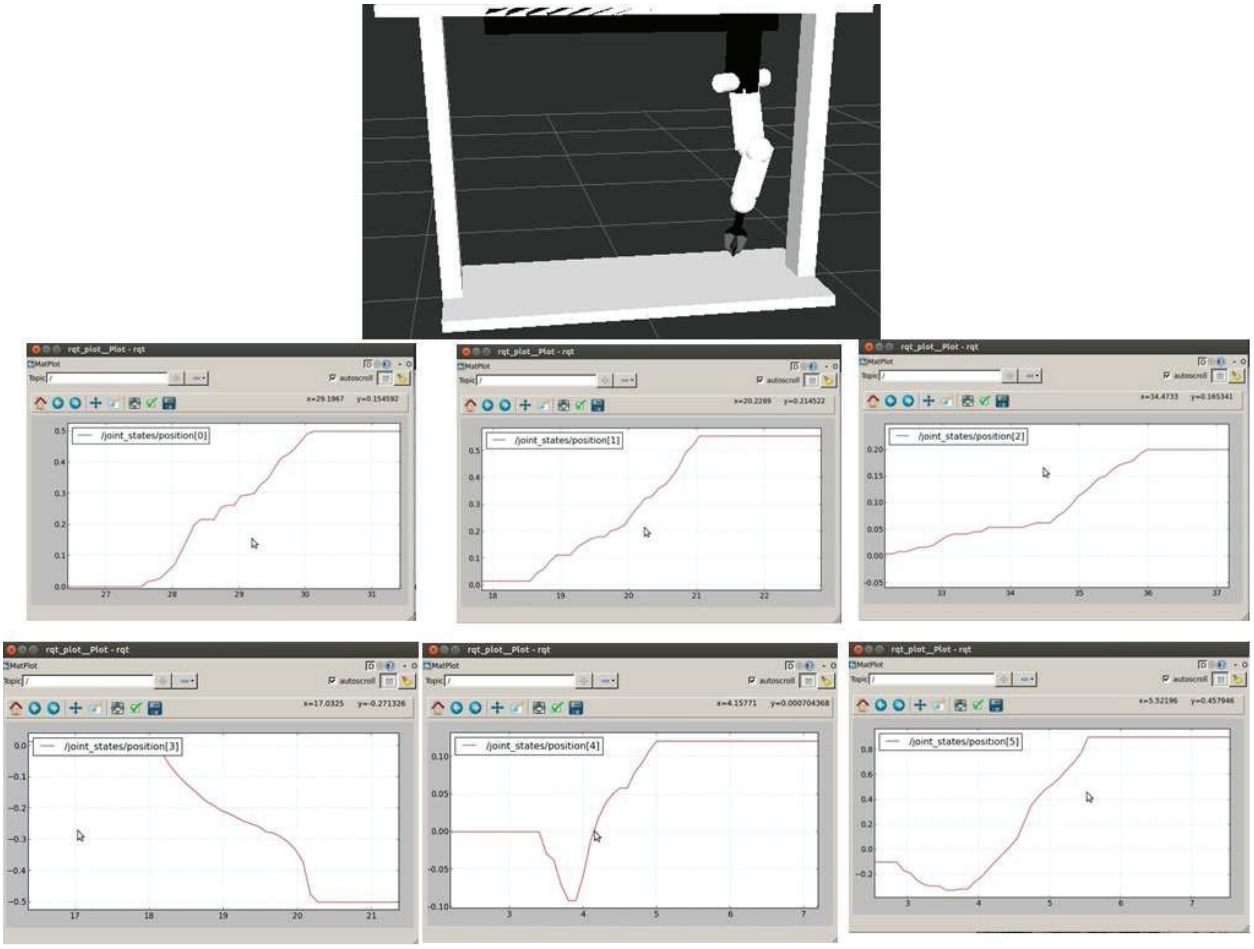



Figure 4.7: Feedback signal simulation

## 4.7 Summary

This chapter covered all experiments that have been done, Disassembling the old controller to test motors and feedback signals from the encoder. Then run an experiment to show how ROS and *LabVIEW* works together. Finally, a path planning experiment in the simulated robot design in ROS.

# Chapter 5

# Conclusion

The purpose of this project is to make the CRS-Plus A150 compatible with modern controller and development systems such as: Robot Operating System and National instruments *LabVIEW*, to decrease the burden of learning proprietary programming languages that serves only limited number of robot models.

The reverse engineering of the CRS-Plus robot arm took much time and effort As with any complex system, and especially old systems, the reverse engineering of the CRS-Plus robot is a time-consuming task, mainly due to the lack of documentation. Once all the components and connections of the robot were identified, several experiments were performed to test the motors, feedback signal and the new controller. The testing of the overall system, namely the full connection of the robot motors and feedback encoders to the controller system developed here, has not been tested. This will be done as future work. When completed, the developed setup will allow the current system setup allow the user to control one motor in the robot arm and that give a motivation to make the other motors work as well. Future research can be done in making the system or the robot demonstrate a complete experiment either in ROS or *LabVIEW*

# Appendix: URDF Code

```
<?xml version="1.0"?>

<robot name="isu">

<link name="base_link">

    <visual>

      <geometry>

        <box size="0.7112 1.7526 .05"/>

      </geometry>

      <origin rpy="0 0 0" xyz="0 0 0"/>

      <material name="white">

        <color rgba="1 1 1 1"/>

      </material>

    </visual>

</link>


<link name="right_leg">

    <visual>

      <geometry>

        <box size="0.1778 0.0889 1.3"/>

      </geometry>
```

```
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <material name="white">
        <color rgba="1 1 1 1"/>
      </material>
    </visual>
  </link>

  <joint name="base_to_right_leg" type="fixed">
    <parent link="base_link"/>
    <child link="right_leg"/>
    <origin xyz="0 -.836 .63"/>
  </joint>

<link name="left_leg">
    <visual>
      <geometry>
        <box size="0.1778 0.0889 1.3"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <material name="white">
        <color rgba="1 1 1 1"/>
      </material>
    </visual>
```

```
  </link>


  <joint name="base_to_left_leg" type="fixed">
    <parent link="base_link"/>
    <child link="left_leg"/>
    <origin xyz="0 .836 .63"/>
  </joint>


<link name="roof">
    <visual>
      <geometry>
        <box size="0.1778 2.1 .04572"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <material name="white">
        <color rgba="1 1 1 1"/>
      </material>
    </visual>
  </link>


  <joint name="roof_to_left_leg" type="fixed">
    <parent link="left_leg"/>
    <child link="roof"/>
```

```
    <origin xyz="0 -.836 0.63"/>
  </joint>

<link name="black_drive">
    <visual>
      <geometry>
        <box size="0.1778 1.2446 .1016"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <material name="black">
        <color rgba="0 0 0 1"/>
      </material>
    </visual>
  </link>

  <joint name="roof_to_black_drive" type="fixed">
    <parent link="roof"/>
    <child link="black_drive"/>
    <origin xyz="0 0 -0.043"/>
  </joint>

<joint name="prismatic_joint" type="prismatic">
    <parent link="black_drive"/>
```

```
    <child link="mani_base"/>
    <limit effort="1000.0" lower="0.5" upper="-0.5" velocity="0.5"/>
    <origin rpy="0 0 1.57" xyz="0 0 -.09"/>
  </joint>

  <link name="mani_base">
    <visual>
      <geometry>
        <cylinder length="0.1524" radius=".0762"/>
      </geometry>
      <origin rpy="0 0 0 " xyz="0 0 0"/>
      <material name="Black">
        <color rgba="0 0 0 1"/>
      </material>
    </visual>
  </link>



<joint name="1st_joint" type="revolute">
    <parent link="mani_base"/>
    <child link="1st_link"/>
    <limit effort="1000.0" lower="-0.9" upper="0.9" velocity="0.5"/>
```

```
    <origin rpy="0 -1.57 1.57" xyz="0 0 -0.15"/>
  </joint>


  <link name="1st_link">
    <visual>
      <geometry>
        <cylinder length="0.1524" radius=".0762"/>
      </geometry>
      <origin rpy="0 1.57 0 " xyz="0 0 -0.003"/>
      <material name="black">
        <color rgba="0 0 0 1"/>
      </material>
    </visual>
  </link>
<link name="handle">
    <visual>
      <geometry>
        <cylinder length="0.3048" radius=".04"/>
      </geometry>
      <origin rpy="1.57 0 0" xyz="-0.03 0 0.05"/>
      <material name="grey">
        <color rgba="1 1 1 1"/>
      </material>
```

```
    </visual>
  </link>

  <joint name="1st_link_to_handle" type="fixed">
    <parent link="1st_link"/>
    <child link="handle"/>
    <origin xyz="0 0 -0.05"/>
  </joint>

<joint name="2nd_joint" type="revolute">
    <parent link="1st_link"/>
    <child link="2nd_link"/>
    <limit effort="1000.0" lower="0.5" upper="-0.5" velocity="0.5"/>
    <origin rpy="0 0 1.57" xyz="0 0 0"/>
  </joint>

  <link name="2nd_link">
    <visual>
      <geometry>
        <box size="0.2694 0.1016 .0889"/>
      </geometry>
      <origin rpy="0 0 -1.57 " xyz="0 0.2 0"/>
      <material name="grey">
```

```
        <color rgba="1 1 1 1"/>
      </material>
    </visual>
  </link>

<link name="2nd_link_sphere1">
    <visual>
      <geometry>
        <sphere radius="0.06"/>
      </geometry>
      <material name="white"/>
    </visual>
  </link>

  <joint name="2nd_design1" type="fixed">
    <parent link="2nd_link"/>
    <child link="2nd_link_sphere1"/>
        <origin xyz="0 0.32 0"/>
  </joint>

<joint name="3rd_joint" type="revolute">
    <parent link="2nd_link"/>
    <child link="3rd_link"/>
```

```
    <limit effort="1000.0" lower="0.5" upper="-0.5" velocity="0.5"/>
    <origin rpy="0 0 0" xyz="0 0.43 0"/>
  </joint>

  <link name="3rd_link">
    <visual>
      <geometry>
        <box size="0.254 0.0916 .0889"/>
      </geometry>
      <origin rpy="0 0 -1.57 " xyz="0 0 0"/>
      <material name="grey">
        <color rgba="1 1 1 1"/>
      </material>
    </visual>
  </link>

<link name="3rd_link_sphere1">
    <visual>
      <geometry>
        <sphere radius="0.06"/>
      </geometry>
      <material name="white"/>
    </visual>
```

```
  </link>


  <joint name="3rd_design1" type="fixed">
    <parent link="3rd_link"/>
    <child link="3rd_link_sphere1"/>
        <origin xyz="0 0.16 0"/>
  </joint>


<link name="3rd_link_sphere2">
    <visual>
      <geometry>
        <sphere radius="0.06"/>
      </geometry>
      <material name="white"/>
    </visual>
  </link>


  <joint name="3rd_design2" type="fixed">
    <parent link="3rd_link"/>
    <child link="3rd_link_sphere2"/>
        <origin xyz="0 -0.1 0"/>
  </joint>
```

```
<joint name="4th_joint" type="revolute">
    <parent link="3rd_link"/>
    <child link="4th_link"/>
    <limit effort="1000.0" lower="0.5" upper="-0.5" velocity="0.5"/>
    <origin rpy="0 0 0" xyz="0 0.17 0"/>
  </joint>

  <link name="4th_link">
    <visual>
      <geometry>
        <cylinder length="0.1524" radius=".02"/>
      </geometry>
      <origin rpy="1.57 0 0 " xyz="0 0.07 0"/>
      <material name="black">
        <color rgba="0 0 0 1"/>
      </material>
    </visual>
  </link>

<joint name="5th_joint" type="revolute">
    <parent link="4th_link"/>
    <child link="5th_link"/>
```

```
    <limit effort="1000.0" lower="0.9" upper="-1.1" velocity="1.5"/>
    <origin rpy="0 0 1.57" xyz="0 0.15 0"/>
  </joint>


  <link name="5th_link">
    <visual>
      <geometry>
        <cylinder length="0.01" radius=".04"/>
      </geometry>
      <origin rpy="0 1.57 0 " xyz="0 0 0"/>
      <material name="black">
        <color rgba="0 0 0 1"/>
      </material>
    </visual>
  </link>
<joint name="left_gripper_joint" type="revolute">
    <axis xyz="0 0 1"/>
    <limit effort="1000.0" lower="0.0" upper="0.548" velocity="0.5"/>
    <origin rpy="0 0 0" xyz="0.01 0.01 0"/>
    <parent link="5th_link"/>
    <child link="left_gripper"/>
  </joint>
```

```
<link name="left_gripper">
  <visual>
    <origin rpy="0.0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://pr2_description/meshes/gripper_v0/l_finger.dae"/>
    </geometry>
  </visual>
</link>

<joint name="left_tip_joint" type="fixed">
  <parent link="left_gripper"/>
  <child link="left_tip"/>
</joint>

<link name="left_tip">
  <visual>
    <origin rpy="0.0 0 0" xyz="0.09137 0.00495 0"/>
    <geometry>
      <mesh filename="package://pr2_description/meshes/gripper_v0/l_finger_tip.dae
    </geometry>
  </visual>
</link>
```

```
<joint name="right_gripper_joint" type="revolute">
  <axis xyz="0 0 -1"/>
  <limit effort="1000.0" lower="0.0" upper="0.548" velocity="0.5"/>
  <origin rpy="0 0 0" xyz="0.01 -0.01 0"/>
  <parent link="5th_link"/>
  <child link="right_gripper"/>
</joint>

<link name="right_gripper">
  <visual>
    <origin rpy="-3.1415 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://pr2_description/meshes/gripper_v0/l_finger.dae"/>
    </geometry>
  </visual>
</link>

<joint name="right_tip_joint" type="fixed">
  <parent link="right_gripper"/>
  <child link="right_tip"/>
</joint>

<link name="right_tip">
```

```
    <visual>
      <origin rpy="-3.1415 0 0" xyz="0.09137 0.00495 0"/>
      <geometry>
        <mesh filename="package://pr2_description/meshes/gripper_v0/l_finger_tip.dae
      </geometry>
    </visual>
  </link>



</robot>
```

# References


[1] Robert Ayres and Steven Miller. Industrial robots on the line. *Technology review*, 85:34–36, 1982.

[2] Herman Bruyninckx. Open robot control software: the orocos project. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 3, pages 2523–2528. IEEE, 2001.

[3] Stefano Carpin, Michael Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. Usarsim: a robot simulator for research and education. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1400–1405. IEEE, 2007.

[4] Inc. Clearpath Robotics. Overview of the labview-ros toolkit. In *Clearpath Robotics*, pages 1–8, May 2012.

[5] Martyn Cooper, David Keating, William Harwin, and Kerstin Dautenhahn. Robots in the classroom-tools for accessible education. *Assistive Technology on the Threshold of the New Millennium*, pages 448–452, 1999.

[6] CRS. Rapl-3 language reference guide. http://phoenix.goucher.edu/ jillz/cs325robotics/RAPL-3LanguageReferenceGuid e.pdf, Aug 2013. UMI-R3-210.

[7] K. DeMarco, M.E. West, and T.R. Collins. An implementation of ros on the yellowfin autonomous underwater vehicle (auv). In *OCEANS 2011*, pages 1–7, Sept 2011.

[8] K. Goldberg, S. Gentner, C. Sutter, and J. Wiegley. The mercury project: a feasibility study for internet robots. *Robotics Automation Magazine, IEEE*, 7(1):35–40, Mar 2000.

[9] Huosheng Hu, Lixiang Yu, Pui Wo Tsui, and Quan Zhou. Internet-based robotic systems for teleoperation. *Assembly Automation*, 21(2):143–152, 2001.

[10] Jared Jackson. Microsoft robotics studio: A technical introduction. *Robotics & Automation Magazine, IEEE*, 14(4):82–87, 2007.

[11] Tobias Kaupp, Alex Brooks, Ben Upcroft, and Alexei Makarenko. Building a software architecture for a human-robot team using the orca framework. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3736–3741. IEEE, 2007.

[12] A. Kleiner and A. Kolling. Guaranteed search with large teams of unmanned aerial vehicles. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2977–2983, May 2013.

[13] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. Yarp: Yet another robot platform. *International Journal of Advanced Robotic Systems*, 3(1), 2006.

[14] D Montemerlo, Nicholas Roy, and Sebastian Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2436–2441. IEEE, 2003.

[15] Paul Michael Newman. Moos-mission orientated operating suite. *Massachusetts Institute of Technology, Tech. Rep*, 2299(08), 2008.

[16] NI. Compactrio i/o modules. http://sine.ni.com/nips/cds/view/p/lang/en/nid/14147, Aug 2012.

[17] NI. Ni compactrio. http://www.ni.com/compactrio/, Aug 2012.

[18] NI. Introduction to the ni labview fpga module. http://www.ni.com/webcast/240/en/, Des 2013.

[19] NI. Ni labview real-time module. http://www.ni.com/labview/realtime/, Des 2013.

[20] NI. What is labview? http://www.ni.com/newsletter/51141/en/, Aug 2013.

[21] GERRY O'BRIEN. Crsplus robot series promotional poster. http://phoenix.goucher.edu/jillz/cs325robotics/RAPL-3LanguageReferenceGuid e.pdf, Aug 2013.

[22] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.

[23] Martin Saerbeck, Tom Schut, Christoph Bartneck, and Maddy D Janse. Expressive robots in education: varying the degree of social supportive behavior of a robotic tutor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1613–1622. ACM, 2010.

[24] TullyFoote. fuerte/installation/ubuntu. http://wiki.ros.org/fuerte/Installation/Ubuntu, Aug 2012.

[25] TullyFoote. Ros/concepts. http://wiki.ros.org/ROS/Concepts, Des 2013.

[26] Xiao-Gang Wang, M. Moallem, and R.V. Patel. An internet-based distributed multiple-telerobot system. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 33(5):627–634, Sept 2003.

[27] WilliamWoodall. Ros/introduction. http://wiki.ros.org/ROS/Introduction, Oct 2013.

[28] Ning Xi and Tzyh Jong Tarn. Action synchronization and control of internet based telerobotic systems. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 219–224. IEEE, 1999.