2012

Usability of Command Strategies for the Phantom Omni/Schunk 5DOF Teleoperation Setup

Brandon Allen Charles Peters

THE FLORIDA STATE UNIVERSITY

FAMU-FSU COLLEGE OF ENGINEERING

USABILITY OF COMMAND STRATEGIES FOR THE PHANTOM OMNI/SCHUNK 5DOF TELEOPERATION SETUP

By

BRANDON ALLEN CHARLES PETERS

A Thesis submitted to the
Department of Electrical and Computer Engineering
In partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Spring Semester, 2012

Brandon Allen Charles Peters defended this thesis on March 15, 2012.

The members of the supervisory committee were:

Carl A. Moore
Professor Directing Thesis

Rodney G. Roberts
Co-Advisor

Mark H. Weatherspoon
Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the thesis has been approved in accordance with university requirements.

ACKNOWLEDGEMENTS

First and foremost I would like to thank Dr. Carl A. Moore, Jr for giving me the opportunity to work under his wing for completion of my Master's degree. Dr. Moore's guidance and mentorship in the lab, on my research, and in other areas of life were instrumental to my success as a graduate student and person. I would like to thank Dr. Rodney G. Roberts for is guidance on my research, particularly pertaining to equations and robotics, and for getting me involved in my first conference. Dr. Oscar Gonzalez's input on the setup and focus of the experiment was invaluable, and I would like to acknowledge him for that. I would like to acknowledge Dr. Mark Weatherspoon, and thank him for choosing to serve on my committee.

I would like to give praises to my lab colleagues; Dr. Xiaobi Li, Collins Adetu, Shannon Whyms, Twan Capehart, and Kirby Little for the many inputs and suggestions on my research. I also thank the many volunteers who participated in my experiment, for I could not have done it without your willingness to give up some of your time.

Lastly, I would like to thank all my many family members, friends, and other colleagues for their much needed support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Teleoperation is the control, operation, and manipulation of a robot or other device over distance. The way in which the master controller dictates how the slave moves in its workspace is called a command strategy. Here, two command strategies are implemented for a teleoperation setup consisting of a Phantom Omni as the master controller, and a 5-DOF serial manipulator as the remote slave. The forward and inverse kinematics, the Jacobian and inverse Jacobian, for the master controller are shown. The forward and inverse kinematics, and the Jacobian are developed for the serial manipulator which serves as the slave. An overview of the control program with its various components is given. The position-position command strategy and position-speed command strategy and how they are implement for the teleoperation setup is detailed. The smoothing of input values and its importance is discussed and shown in detail. The need for self-collision avoidance and ensuring the remote arm does not strike the table top is discussed, along with the algorithms that govern each type of collision avoidance. The experiment setup and results are present next, and the result are discussed. Finally, this work concludes with a discussion of future work and considerations.

1 INTRODUCTION

1.1 What is Teleoperation

Teleoperation is the control, operation, and manipulation of a robot or other device over distance. The device used for input is referred to as the master or master controller. The device being controlled by the master to perform a given task is known as the slave or remote device. Any number of devices can serve as a master; in the past, typically keyboard inputs and joysticks were used in telerobotic setups. But within the last several years, other types of master devices are becoming more popular. Haptic input devices are being used more and more because they provide the capability allow the user to receive force feedback about the environment. Hand gestures via computer vision is also emerging as a possible master controller.

1.2 Applications of Teleoperations

Robotic manipulators are playing a bigger role in life altering and life saving medical procedures. A robotic manipulator offers a level of accuracy, precision, and steadiness that simply cannot be matched by any human holding a surgical tool. Also, many medical facilities do not have on staff the necessary medical personnel to perform certain medical procedures. This is where teleoperation has the ability of provide a service to the medical field. If there is no qualified doctor in the area, perhaps one could still be of service, *remotely*. A robot appropriately outfitted with the right tools and instruments could allow a qualified doctor to operate on the patient from a geographically separate location through a teleoperated robotic surgical system. The doctor operates the input device on his/her end, the commands and data is sent over a communication link to the slave side, where the slave device moves appropriately. Typically, the doctor receives feedback data in the form of video and audio to a computer, and force feedback is fed to the master controller.

Teleoperated robots have also made their mark in ocean exploration. Remotely operated underwater serial manipulators mounted on unmanned underwater vehicles (UUV) help scientists and engineers aid in the process of exploring and or manipulating environments man cannot go. Robot is either connected via a long power and communication tether to a vessel on the surface, or communicates wirelessly with scientists, engineers, and staff on the surface who command the robot through a master controller. The master controller can be a separate device interfaced with a computer or just a computer. These robots are used to collect data on all sorts of things ranging from sea bed soil samples, deep sea life, water temperature, salinity, silt, etc. These robots have also been utilized in underwater construction, where structures being built simply could not be done by human divers. Also, the recent BP oil spill saw UUVs used in sealing up the leaking underwater pipe which spewed tons of crude oil.

Teleoperated robots are also very useful for handling toxic materials. Normally, humans in special hazmat suits pick up and move the materials to different locations. However, there are materials too toxic and radioactive to be handed by people even in hazmat suits. Teleoperated robots are suited for this task. A human operator with knowledge of what needs to be done can sit a safe distance away in a control room and perform the clean up. Using a master controller interfaced with a computer, and a manipulator and/or manipulator-mobile ground vehicle combination with camera, the user can survey the environment, decide what to manipulate, and handle the toxic materials without ever having to wear a hazmat suit.

1.3 Similar Work in Command Strategies for Teleoperation

In [13], reseachers developed a telerobotic system for assisting patience with everyday tasks essential to living. Their system consisted of Phantom Omni haptic device for the master controller and a 6-DOF Puma 560 coupled with a Barrett Technologies 4-DOF gripper for the slave device. This system contains four teleoperation control modes, two were based on laser guidance of the remote arm in the workspace while the other two were modes based strictly on the motion of the master controller and the laser was not used in these modes. The latter two control modes from [13] are relevant to this paper. The first was a position-based teleoperation mode using differential translations and differential rotations to command the translation and rotation of the remote arm. The other non-laser teleoperation mode was a velocity-based mode in which the the differential translations and rotations were used to compute the speed and direction of the remote arm. Like a joystick, the initial position acted like the center, so the farther away the master controller moved from the center the faster the remote arm would move in that direction.

Teleoperation also entails the control and command of remote mobile or wheeled robots. The contribution of [4] was a hybrid between the position-position and position-speed command strategy for motion control. Here, the researchers used their method to command the motion of a remote wheeled robot through a maze. The master controller was a Phantom Premium 1.5A and the remote robot was a Pioneer 3-DX. For their experiments, they first tested the position-speed strategy in which position of the master controller was multiplied by a constants matrix to command the speed at which the robot would move. The position-position strategy used a different constants matrix to command the position of the robot in the maze. The hybrid strategy from [4] came in two versions, manual or automatic switching. The manual option allowed the user to decide when they wanted to switch between position-position and position-speed. The automatic option utilized the robot's sensors to switch between the command strategies, using position-position command when the master controller moved slowly or the remote robot was within a predetermined range of an obstacle or object. Otherwise the hybrid strategy relied on position-speed command.

1.4 PURPOSE OF WORK

In this work, the attempt is to implement and test two command strategies for a telerobotic system consisting of one master controller and a remote serial manipulator. The position-position command strategy implemented is very common, however, the position-speed command is somewhat unconventional. This type of position-speed command mode will allow the user less of the total volume of the master controller's workspace, while still allowing the remote slave to reach most points in its workspace. User testing will be conducted to determine which of the command modes the user think is the easiest to use for this system setup. Finally, we want to suggest which command strategy is better given a particular situation/application.

2 SYSTEM DESCRIPTION

2.1 Typical System Setup

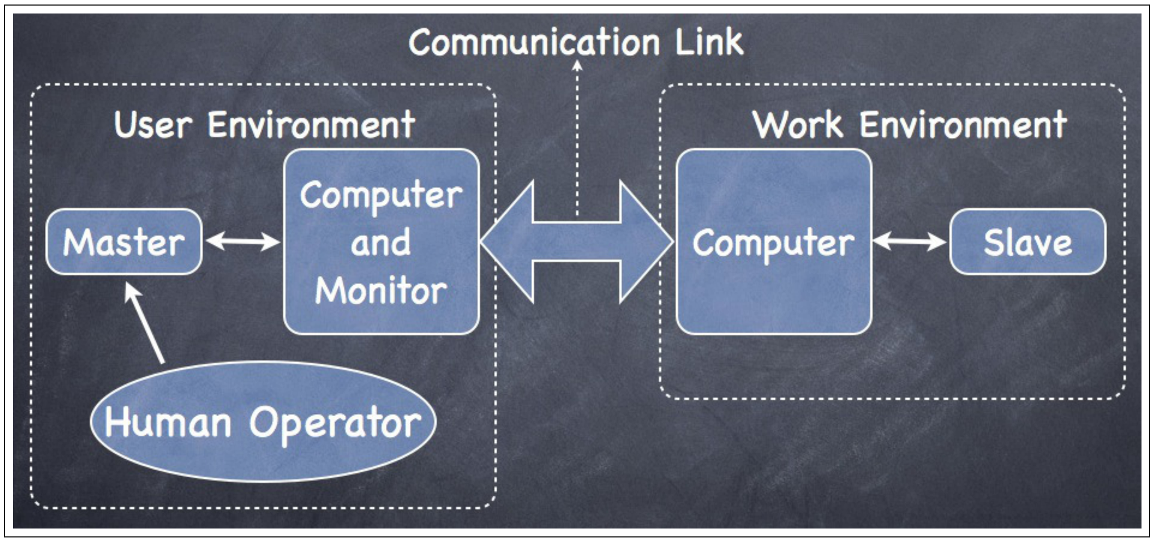


Figure 2.1: Teleoperation system setup

Several key elements are encompassed in a teleoperation system. There is a master controller, an operator, computer, communication link, and a slave device. The master controller, operator, and a computer normally exist in the user environment. The master device is linked to a computer which normally runs the computer program to control or read data from the the master, which is manipulated by the human operator. The slave environment comprises of the slave device, and sometimes a computer to which the slave is linked. The communication link connects the user and slave environments, allowing data to travel from one workspace to the other, with the link being connected to the computers of both environments. In cases where there is no computer on the slave side, the communication link is connected directly to the slave, as is the case in our setup.

2.2 System Setup for This Work

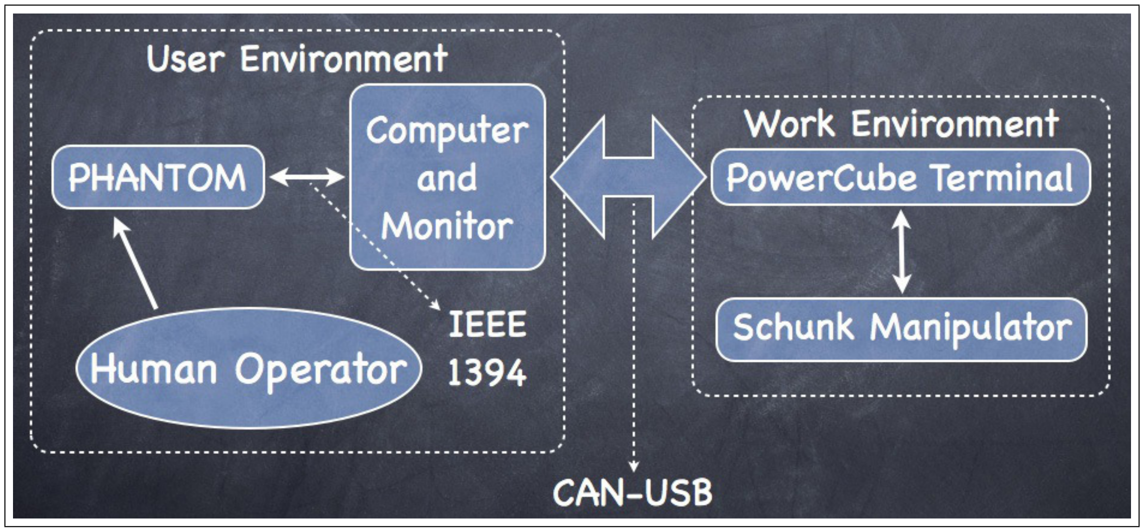


Figure 2.2: Our teleoperation system setup

Our system mostly comprises of the previously mentioned components. In the user environment, there is a master controller, and a computer. In the slave environment, there is only the slave manipulator and its DC power supply. In our setup, the communication link is connected on one end to the computer in the user space and directly to the slave device on the other end.

3 MASTER CONTROLLER

3.1 Device Overview

The device which serves as the master controller in our system is the Phantom Omni haptic device produced by SensAble Technologies. This device has a total of 6-DOFs. Three DOFs for the armature, or position part, and three for the gimbal, or the orientation part. Additionally, there is a stylus attached to the end-effector, which has two buttons.

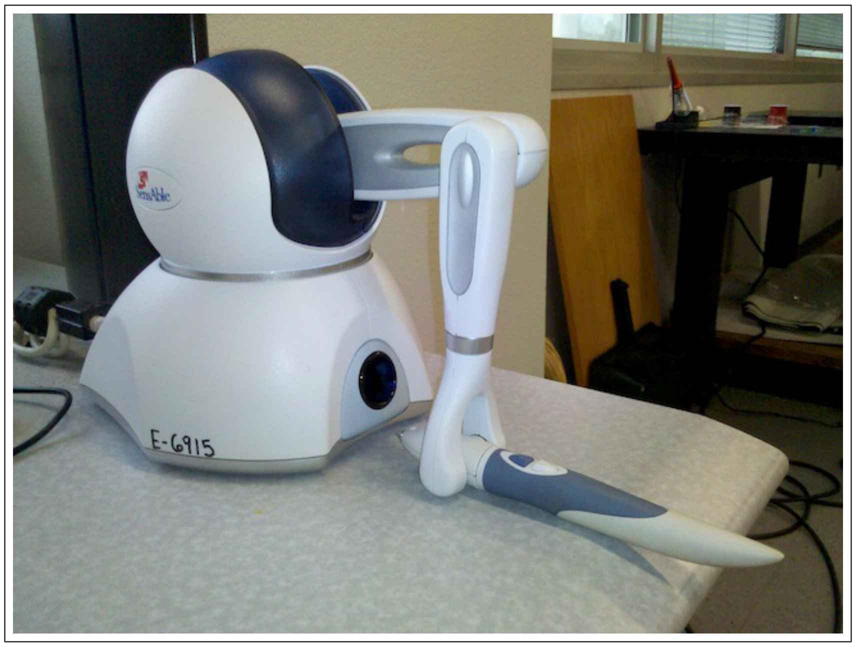


Figure 3.1: The Phantom Omni by Sensable

The Phantom Omni is a very commonly used device in literature. Veras et. al. [13], used the Omni as a master device in a teleoperation setup with a Puma 560 robotic arm to form a system to assist disabled patients with daily tasks. Song et. al [11], used the as a master device to control a 6-DOF robt arm in the tele-rehabilitation system. The Omni was also used as an input device to control a hydraulic machine. Hayn et. al [5], developed a control method where the Omni served as the master controller for the boom and stick segments of an excavator commonly used in construction.

3.2 Forward Kinematics

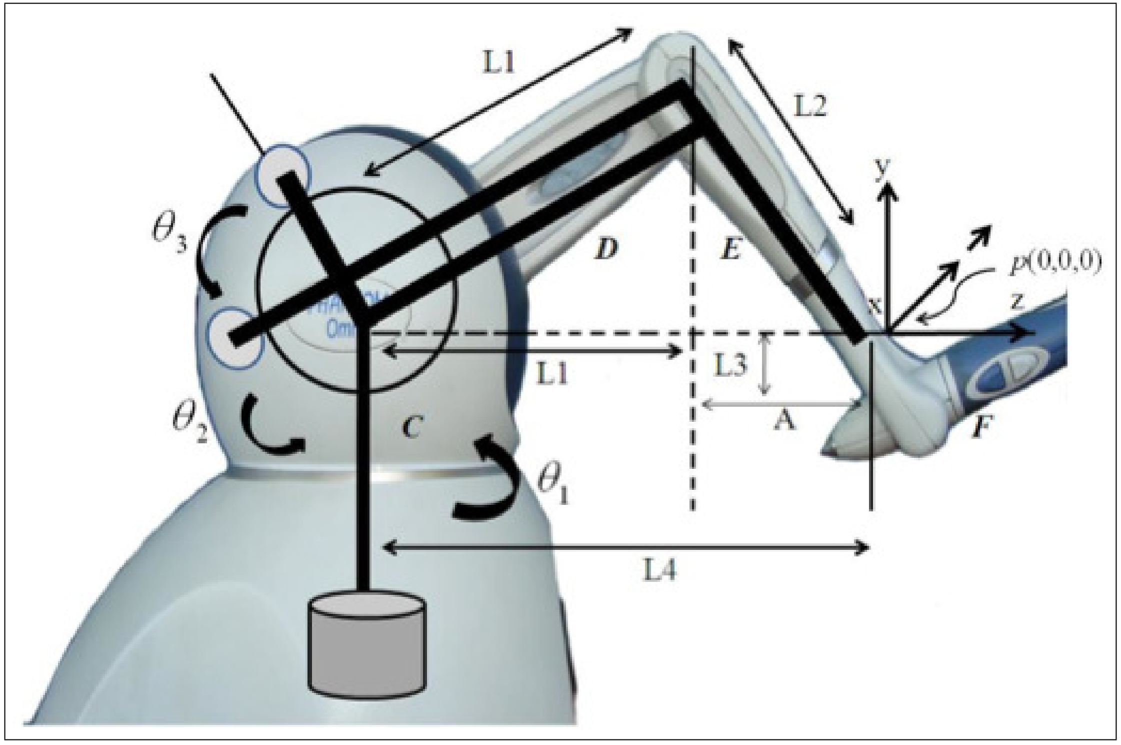


Figure 3.2: Diagram for forward kinematics[6]

The forward kinematics describe the relation relationship transforming joint-space coordinates $\boldsymbol{\theta}$ to task-space coordinates $\boldsymbol{x}$ given joint angles and link lengths and offsets, where $\boldsymbol{\theta}$ and $\boldsymbol{x}$ are vectors. For the Phantom Omni, the forward kinematics are somewhat different in that the origin for the device is not at the base of the mechanism. The manufacturer chose to place the origin at a point *out* in the workspace. Jarillo et. al. [6], published a paper detailing the kinematics and manipulability of the Phantom Omni. Given the joint angles, the equations for a point $P = [\ x \quad y \quad z\ ]^T$ as described by [6] are:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -s_1(L_1c_2 + L_2s_3) \\ L_1s_2 - L_2c_3 + L_3 \\ c_1(L_1c_2 + L_2s_3) - L_4 \end{bmatrix} \tag{3.1}$$

Here, the values for the lengths for the parameters in (3.1) are $L_1 = L_2 = 135.0mm$, $L_3 = 25.0mm$, and $L_4 = 170.0mm$.

3.3 Inverse Kinematics

The inverse kinematics describe the transformation from task-space coordinates $\boldsymbol{x}$ to joint-space coordinates $\boldsymbol{\theta}$, where $\boldsymbol{x}$ and $\boldsymbol{\theta}$ are vectors. For an end effector position $\boldsymbol{x} = [\ x \quad y \quad z\ ]^T$ with know link lengths, the inverse kinematics allow for the computation of the joint angles that contribute to the current end-effector position. Here, a geometric approach is used to determine the inverse kinematics for this master device. From [6], θ_1 is expressed by:

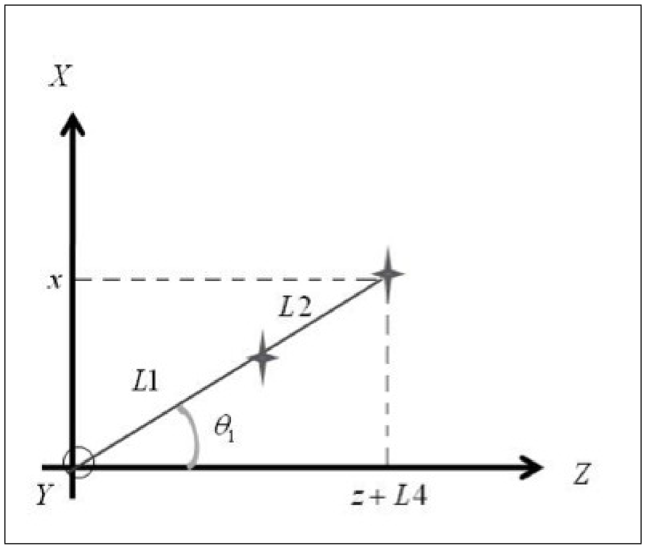


Figure 3.3: Top view, inverse kinematic diagram for the Omni[6]

$$\theta_1 = -atan2(x, z + L_4) \tag{3.2}$$

Next, we wish to find equations for θ_2 and θ_3. The following diagram will be used to help find these equations:

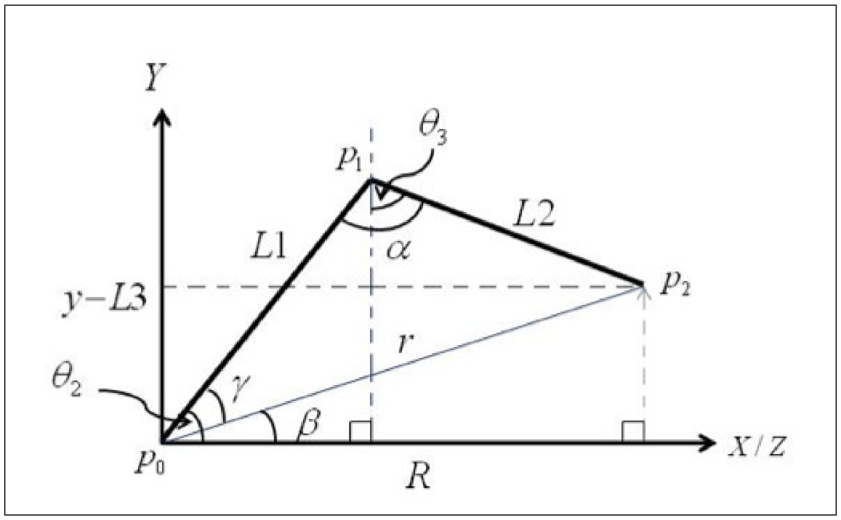


Figure 3.4: Side view, inverse kinematic diagram for the Omni[6]

We solve for the angle β:

$$R = \sqrt{x^2 + (z + L_4)^2} \tag{3.3}$$

$$\beta = atan2(y - L_3, R) \tag{3.4}$$

The equation for the angle γ is determined by applying the Law of Cosines to the triangle $\triangle P_0P_1P_2$:

$$r = \sqrt{x^2 + (z + L_4)^2 + (y - L_3)^2} \tag{3.5}$$

$$\begin{aligned} L_2^2 &= L_1^2 + r^2 - 2L_1 r cos(\gamma) \\ cos(\gamma) &= \tfrac{L_1^2 + r^2 - L_2^2}{2L_1 r} \\ &\equiv D_1 \end{aligned} \tag{3.6}$$

$$\gamma = atan2(\sqrt{1 - D_1^2}, D_1) \tag{3.7}$$

Knowing γ, θ_2 is expressed as:

$$\theta_2 = \beta + \gamma \tag{3.8}$$

The Law of Cosines is applied again to the triangle $\triangle P_0P_1P_2$ as in [6] to solve for α:

$$\begin{aligned} r^2 &= L_1^2 + L_2^2 - 2L_1L_2 cos(\alpha) \\ cos(\alpha) &= \tfrac{L_1^2 + L_2^2 - r^2}{2L_1L_2} \\ &\equiv D_2 \end{aligned} \tag{3.9}$$

$$\alpha = atan2(\sqrt{1 - D_2^2}, D_2) \tag{3.10}$$

$$\theta_3 = \theta_2 + \alpha + \frac{\pi}{2} \tag{3.11}$$

3.4 The Jacobian

The Jacobian shows the relationship between the cartesian end-effector velocity and the joint space angular velocity. The Jacobian is a matrix comprising of first-order partial derivatives of the forward kinematics position equations. Given joint-space speeds, the task-space speeds can be determined via matrix multiplication.

$$\dot{\boldsymbol{x}} = J\dot{\boldsymbol{\theta}} \tag{3.12}$$

For the Phantom Omni, $\boldsymbol{x} \in R^{3\times1}$ is the end effector velocities vector, $J \in R^{3\times3}$ is the Jacobian, and $\dot{\boldsymbol{\theta}} \in R^{3\times1}$ is the angular velocities vector. As determined in [6], the terms of the matrix J are:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -c_1(L_1c_2 + L_2s_3) & L_1s_1s_2 & -L_2s_1c_3 \\ 0 & L_1c_2 & L_2s_3 \\ -s_1(L_1c_2 + L_2s_3) & -L_1c_1s_2 & L_2c_1c_3 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \tag{3.13}$$

3.5 The Inverse Jacobian

The inverse Jacobian allows for the computation of the angular velocity, given the end-effector velocity. Unfortunately, the Phantom Omni's API only allows for the user to query the device for the end-effector velocity.

$$\dot{\boldsymbol{\theta}} = J^{-1}\dot{\boldsymbol{x}} \tag{3.14}$$

Here, the dimensions of the $\dot{\boldsymbol{\theta}}$, J^{-1}, and $\dot{\boldsymbol{x}}$ are the same as their forward Jacobian counterparts. For a matrix M, the inverse exists *if and only if* its determinant is non-zero: $det(M) \neq 0$. The inverse is defined as the adjoint of the matrix divided by its determinant:

$$J^{-1} = \frac{adj(J)}{det(J)} \tag{3.15}$$

If the $det(J) \neq 0$, we can calculate $G = J^{-1}$:

$$G = \begin{bmatrix} \frac{-c_1}{L_1c_2+L_2s_3} & 0 & \frac{-s_1}{L_1c_2+L_2s_3} \\ \frac{s_1s_3}{L_1c_{2-3}} & \frac{c_3}{L_1c_{2-3}} & \frac{c_1s_3}{L_1c_{2-3}} \\ \frac{-c_2s_1}{L_2c_{2-3}} & \frac{c_2}{L_1c_{2-3}} & \frac{c_1c_2}{L_2c_{2-3}} \end{bmatrix} \tag{3.16}$$

4 SLAVE DEVICE

4.1 Device Overview

The slave device for this system is a custom built-to-order 5 DOF serial manipulator manufactured and developed by Schunk. The base joint and second joint are PRL-100 modules, the third and fourth joints are PRL-80 modules, and the fifth joint is a PRL-60 module. The PRL-100 modules are the largest modules and most powerful of the units, followed by the PRL-80s, and the PRL-60s. In addition, there is a linear gripper module attached to the fifth rotary module. In between the linear gripper and fifth rotary module, there is a 3-axis force sensor. The manipulator requires a constant power source, 24V and 21-27A to operate.

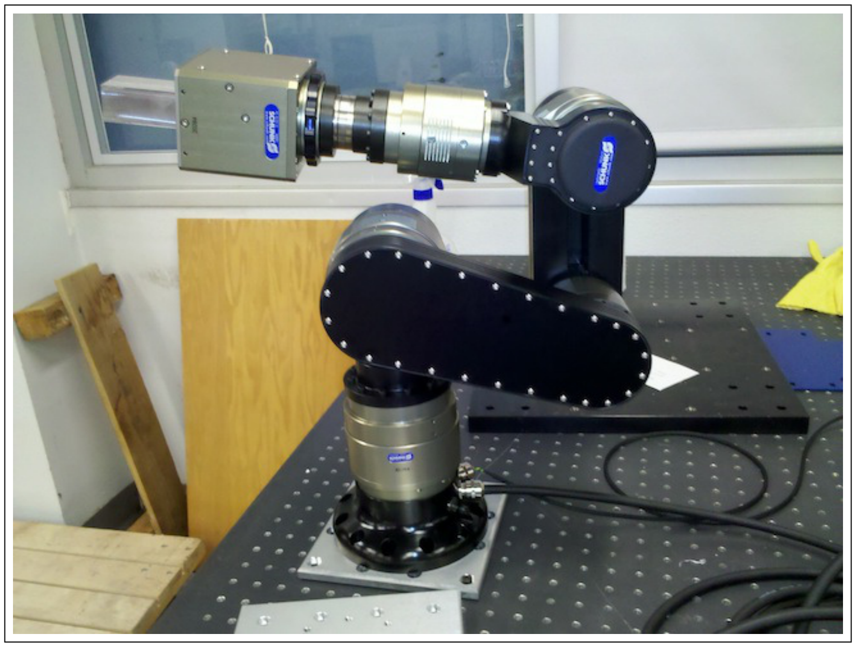


Figure 4.1: 5-DOF serial manipulator from Schunk

4.2 Forward Kinematics

The forward kinematics transform joint-space coordinates $\boldsymbol{\theta}$ to task-space coordinates $\boldsymbol{x}$ given joint angles and link lengths and offsets, where $\boldsymbol{\theta}$ and $\boldsymbol{x}$ are vectors. The most common way to perform forward kinematic analysis for serial manipulators is the Denavit-Hartenberg Convention[12]. The following diagram and table show the frame assignments and DH table, respectively [8]:

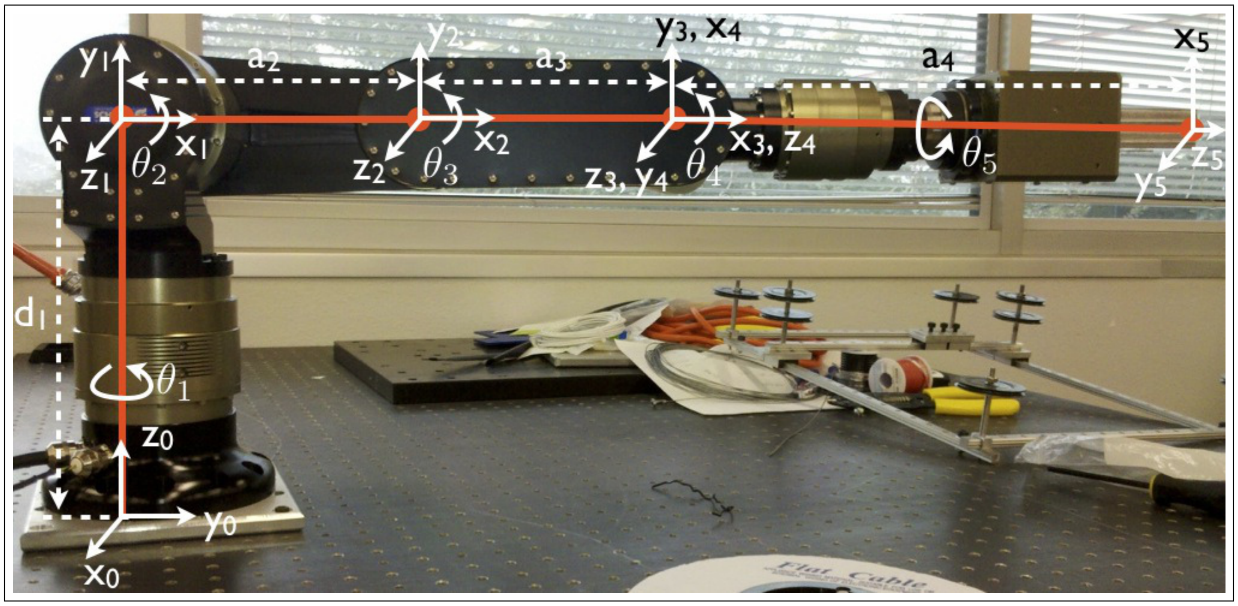


Figure 4.2: Forward kinematic diagram of the manipulator

Table 4.1: DH-Table for frame assignments

Link	a_i	α_i	d_i	θ_i
1	0	$-90°$	d_1	θ_1^*
2	a_2	0	0	θ_2^*
3	a_3	0	0	θ_3^*
4	a_4	$90°$	0	θ_4^*
5	0	0	0	θ_5^*

where a_i are the link lengths, the twist angles α_i, the link offsets d_i, and the joint angles

θ_i. Here, $d_1 = 268.0mm$, $a_2 = 190.0mm$, $a_3 = 170.0mm$, and $a_4 = 362.3mm$. The five homogeneous transformation matrices given by the DH-Table are:

$$A_1 = \begin{bmatrix} C_1 & 0 & -S_1 & 0 \\ S_1 & 0 & C_1 & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

$$A_2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

$$A_3 = \begin{bmatrix} C_3 & -S_3 & 0 & a_3C_3 \\ S_3 & C_3 & 0 & a_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$$A_4 = \begin{bmatrix} C_4 & 0 & S_4 & a_4C_4 \\ S_4 & 0 & -C_4 & a_4S_4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

$$A_5 = \begin{bmatrix} C_5 & -S_5 & 0 & 0 \\ S_5 & C_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

Multiplying the transformation matrices A_1, A_2, A_3, A_4, and A_5 together gives the forward kinematics transformation from the base of the manipulator to the tip of the end-effector [8]:

$$\begin{aligned} T_0^5 &= A_1A_2A_3A_4A_5 \\ &= \begin{bmatrix} c_1c_{234}c_5 - s_1s_5 & -c_1c_{234}s_5 - s_1s_5 & c_1s_{234} & c_1(a_2c_2 + a_3c_{23} + a_4c_{234}) \\ s_1c_{234}c_5 + c_1c_5 & -s_1c_{234}s_5 + c_1c_5 & s_1s_{234} & s_1(a_2c_2 + a_3c_{23} + a_4c_{234}) \\ -s_{234}c_5 & s_{234}s_5 & c_{234} & d_1 - a_2s_2 - a_3s_{23} - a_4s_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{4.6}$$

4.3 Inverse Kinematics

The inverse kinematics transform task-space coordinates $\boldsymbol{x}$ to joint-space coordinates $\boldsymbol{\theta}$ given joint angles and link lengths and offsets, where $\boldsymbol{x}$ and $\boldsymbol{\theta}$ are vectors. For the serial manipulator in this system, a geometric approach was used to analytically determine the inverse kinematic equations. Given a position $\boldsymbol{x}$ and an approach angle θ_d, the inverse kinematics are computed by [8]:

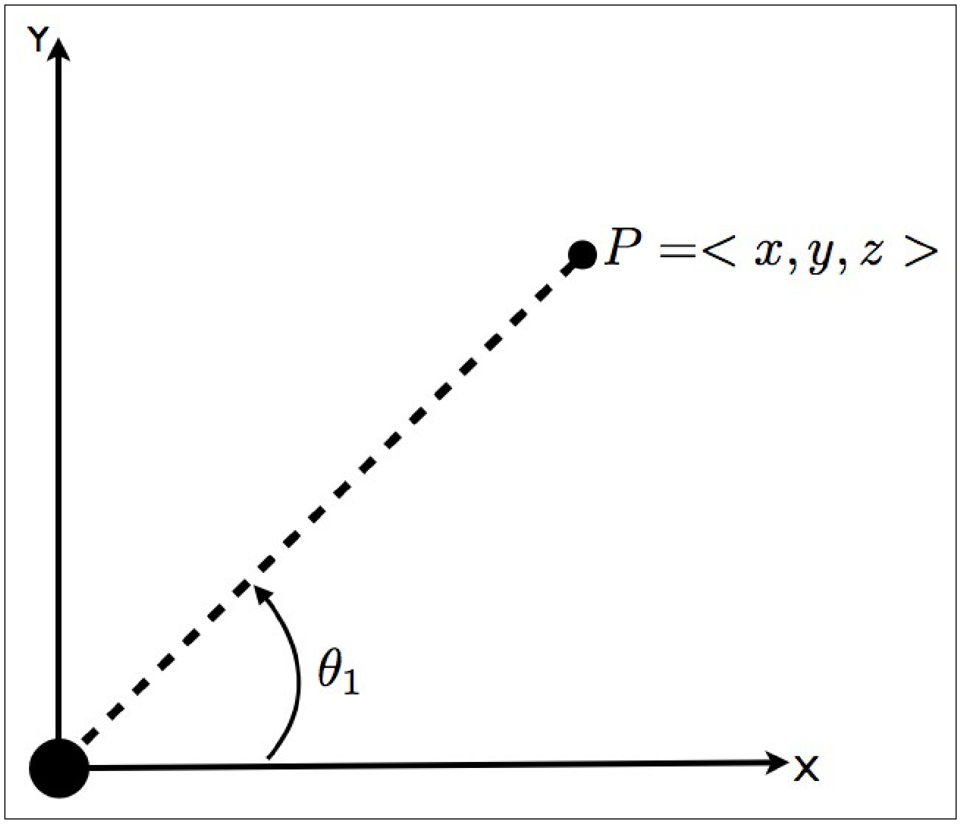


Figure 4.3: Top view of the manipulator

$$\theta_1 = atan2(y, x) \tag{4.7}$$

The following figure was formed to solve for θ_2, θ_3, and θ_4:

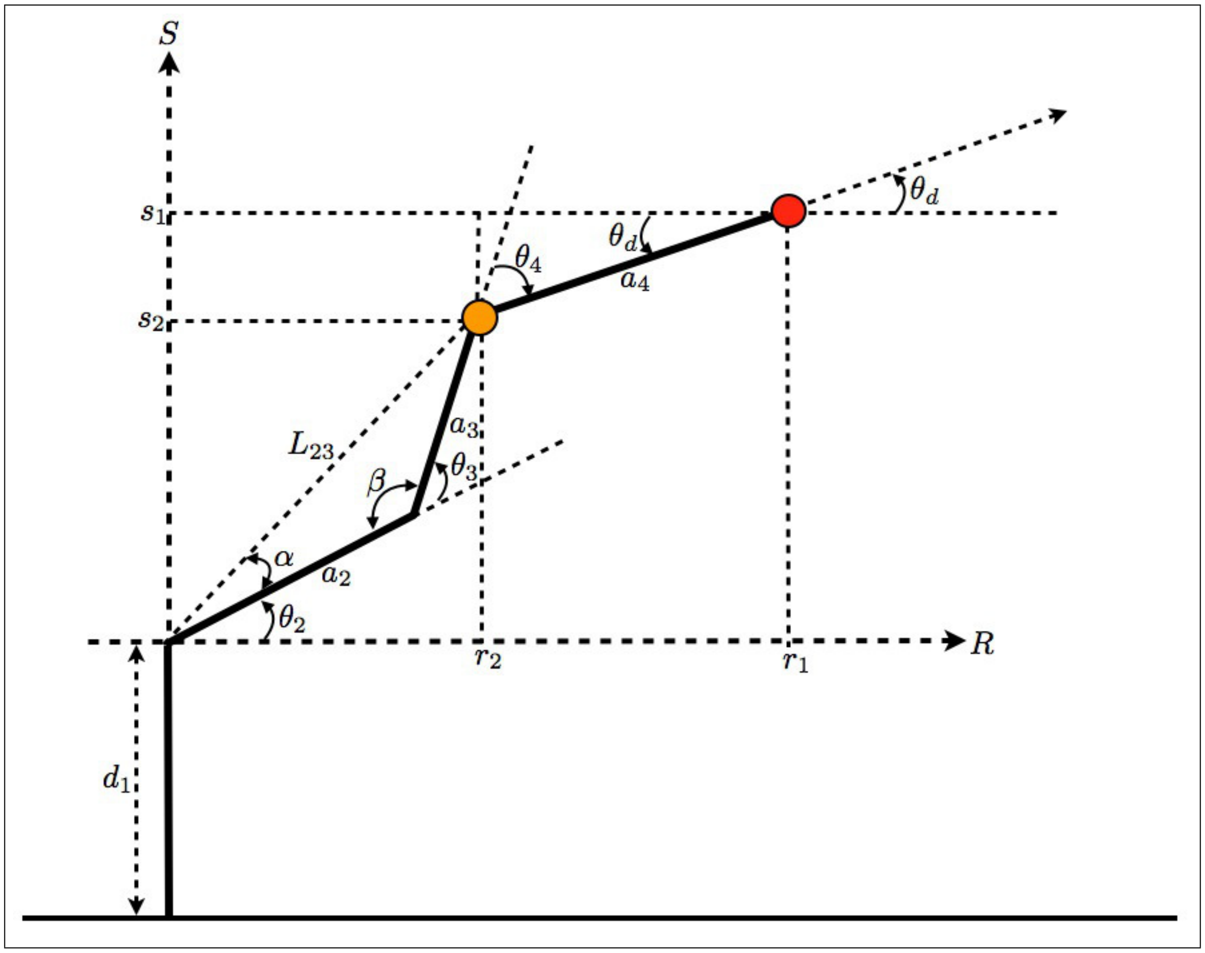


Figure 4.4: Side view, inverse kinematic diagram for manipulator

$$\begin{aligned} r_1 &= \sqrt{x^2 + y^2} \\ s_1 &= z - d_1 \end{aligned} \tag{4.8}$$

Next, θ_d and a_4 are used to track back from the tip of the gripper to the wrist center:

$$\begin{aligned} r_2 &= r_1 - a_4 cos(\theta_d) \\ s_2 &= s_1 - a_4 sin(\theta_d) \end{aligned} \tag{4.9}$$

To solve for θ_3, we employ the Law of Cosines to solve for β first:

$$\begin{aligned} L_{23}^2 &= a_2^2 + a_3^2 - 2a_2a_3cos(\beta) \\ cos(\beta) &= \tfrac{a_2^2+a_3^2-L_{23}^2}{2a_2a_3} \equiv D \end{aligned} \tag{4.10}$$

Here, the variable $L_{23} = \sqrt{r_2^2 + s_2^2}$ is the length from the tail of link a_2 to the head of link a_3, thus forming a triangle. Using the fact that $cos^2(\beta) + sin^2(\beta) = 1$ leads to:

$$sin(\beta) = \sqrt{1 - cos(\beta)^2} = \sqrt{1 - D^2} \tag{4.11}$$

Now, we compute β as

$$\beta = atan2(\sqrt{1 - D^2}, D) \tag{4.12}$$

Using β, we can now obtain θ_3:

$$\theta_3 = \pm(\pi - \beta) \tag{4.13}$$

It is important to know whether the second and third links are in an elbow-up (negative solution) or elbow-down (positive solution) configuration, as it affects the determination of the value of θ_3. Computing the second joint angle requires that we first determine the angle α inside the triangle formed by a_2, a_3, and L_{23}:

$$\alpha = atan2(a_3 s_3, a_2 + a_3 c_3) \tag{4.14}$$

Using the above angle, we can now compute the second joint angle:

$$\theta_2 = atan2(s_2, r_2) - \alpha \tag{4.15}$$

Using the fact that $\theta_d = \theta_2 + \theta_3 + \theta_4$, with three of the angles known, the fourth joint angle is determined by:

$$\theta_4 = \theta_d - (\theta_2 + \theta_3) \tag{4.16}$$

4.4 The Jacobian

The Jacobian is a matrix of first-order partial derivatives that transforms the joint-space speeds into task-space speeds. The Jacobian for the manipulator was analytically determined.

$$\dot{\boldsymbol{x}} = J\dot{\boldsymbol{\theta}} \tag{4.17}$$

Here $\dot{\boldsymbol{x}} \in R^{3\times1}$ is the cartesian end-effector velocity, $J \in R^{3\times4}$ is the Jacobian matrix of the Schunk robot, and $\dot{\boldsymbol{\theta}} \in R^{4\times1}$ is the joint-space velocity vector.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} \\ J_{21} & J_{22} & J_{23} & J_{24} \\ J_{31} & J_{32} & J_{33} & J_{34} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \end{bmatrix} \tag{4.18}$$

Here, each of the forward kinematic equations for position in (4.6) is differentiated with respect to the each joint angle, θ_1, θ_2, θ_3, and θ_4:

$$\begin{aligned}
J_{11} &= -s_1(a_2c_2 + a_3c_{23} + a_4c_{234}) \\
J_{21} &= c_1(a_2c_2 + a_3c_{23} + a_4c_{234}) \\
J_{31} &= 0 \\
J_{12} &= -c_1(a_2s_2 + a_3s_{23} + a_4s_{234}) \\
J_{22} &= -s_1(a_2s_2 + a_3s_{23} + a_4s_{234}) \\
J_{32} &= -a_2c_2 - a_3c_{23} - a_4c_{234} \\
J_{13} &= -c_1(a_3s_{23} + a_4s_{234}) \\
J_{23} &= -s_1(a_3s_{23} + a_4s_{234}) \\
J_{33} &= -a_3c_{23} - a_4c_{234} \\
J_{14} &= -a_4c_1s_{234} \\
J_{24} &= -a_4s_1s_{234} \\
J_{34} &= -a_4c_{234}
\end{aligned} \tag{4.19}$$

4.5 Inverse Jacobian

To find the inverse Jacobian one cannot use a traditional inverse function (i.e., traditional way of inverting a matrix). This is because the matrix J is not $n \times n$. Instead, the Moore-Penrose pseudoinverse must be employed when $det(JJ^T) \neq 0$:

$$J^+ = J^T(JJ^T)^{-1} \tag{4.20}$$

where J^+ is the Moore-Penrose pseudoinverse of the Jacobian matrix J. The PowerCube API provides a function to directly obtain the joint speeds, thus the computation of the matrix J^{-1} is unnecessary.

4.6 The Wire-cutter and Wire Stations

The Schunk serial manipulator in our teleoperaton setup will be used in a wire-cutting experiment. Unfortunately, the standard gripper fingers cannot cut wires on their own, nor are they fit for holding a wire-cutting tool. A special-purpose wire-cutting apparatus was designed and machined to replace the standard fingers for the experiment.

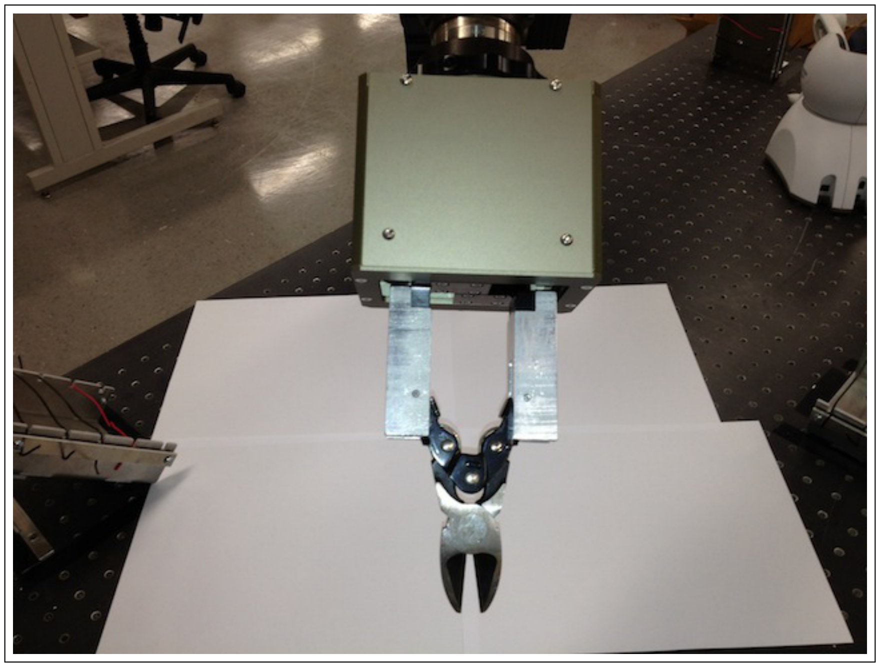


Figure 4.5: Specially designed wire-cutter for 5 DOFs manipulator

There were some design requirements on how the cutter should be built:

- Easy to integrate with current gripper module
- Lightweight and does not add significant size to the robot
- The gripper opens and closes along the center line of the module.

An off-the-shelf pair of wire-cutting pliers was purchased from a local hardware store. The pliers were cut at a position along the length of the handles to allow the Schunk to adequately open and close with the apparatus attached The cutter only adds about $82mm$ of length to the robot's end-effector.

The wire stations house the wires to be used in the experiment. Here is a list of the hard design requirements for the housings:

- Hold five wires firmly in place

- Allows space between wires so the robot can cut one wire without touching nearby wires
- Makes it easy to install and remove wires
- Three stations total, one flat, one angled, and one fully vertical.

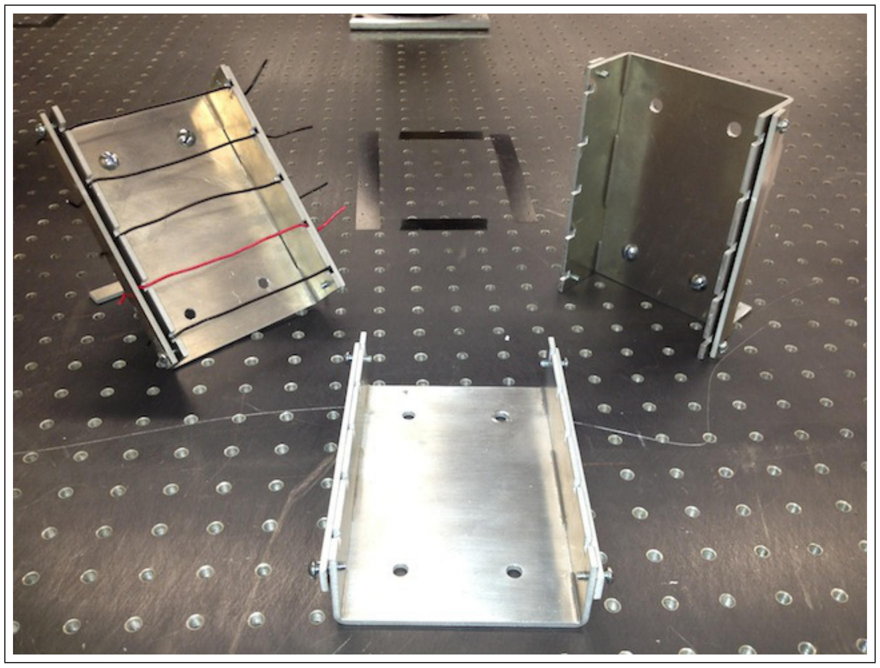


Figure 4.6: Wire stations

The stations are made of light-weight aluminum. The holding mechanisms on each side of a station screw in to hold each of the wires down.

5 CONTROL PROGRAM OVERVIEW

The section will give a rough overview of the program that is responsible for pulling all the aspects of the teleoperation setup together at the code level.

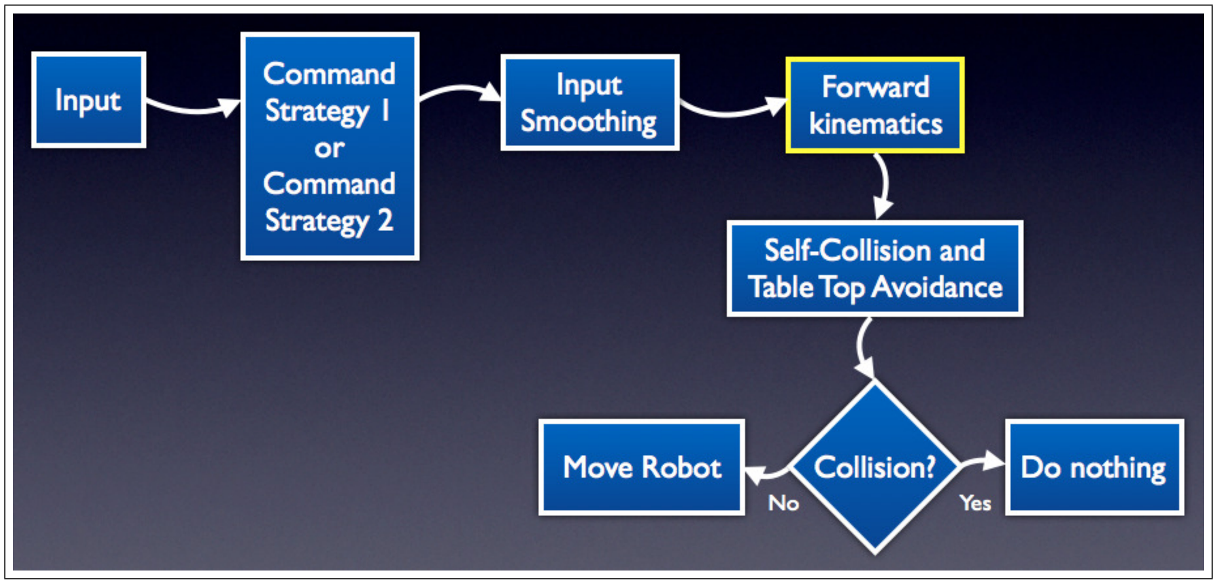

Input
Command Strategy 1 or Command Strategy 2
Input Smoothing
Forward kinematics
Self-Collision and Table Top Avoidance
Collision?
Move Robot
No
Yes
Do nothing

Figure 5.1: Control program diagram

The input data is fed to the control program from the master controller. Then, the input values are processed based on which command strategy is going to be used. The joint values $\hat{\boldsymbol{u}}$ (7.2) are fed into a low-pass filter, which also allows the program to compute the next commanded joint angles $\boldsymbol{q}_{n+1}$ (7.2) for the remote arm before it sends them. Those values are then fed into the forward kinematics to compute the task-space position $\boldsymbol{x}_{n+1}$. If $\boldsymbol{x}_{n+1}$ is low, or the remote arm fails the self-collision test, the input values will be ignored and will not be commanded to the remote arm. Otherwise, the values will be sent to the remote arm and the arm will more to that joint location $\boldsymbol{q}_{n+1}$. Except for the forward kinematics which was covered in a previous section, the modules in the diagram presenting command strategy 1 and command strategy 2, input smoothing, and self-collision and table top avoidance will be covered in more detail in subsequent sections.

6 COMMAND STRATEGIES

6.1 Position-Position Command

In Position-Position Command, or PPC, the position of the master controller's end-effector in its workspace is used to control and command the position of slave device's end-effector in its workspace. Usually, this involves scaling up the displacement of the master and sending it to the slave [4]:

$$\begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} \tag{6.1}$$

where $[x_m \; y_m \; z_m]^T$ is the master's position in its workspace, $K = diag(k_x \; k_y \; k_z)$ is a matrix of constant scaling terms, and $[x_s \; y_s \; z_s]^T$ is the slave's position in its workspace. Scaling is very important in cases where the master and slave workspace volumes are not the same. Typically, the workspace of the master controller is more limited than that of the slave device, which may facilitate the need for large scaling constants [4].

Farkhatdinov et. al [4], used a Phantom Premium 1.5A as a master controller and the position-position command strategy to steer a two-wheel mobile robot through a corridor with designated stopping points along the way. Ahn et. al. [1], also used position-position command for their master-slave setup while investigating input saturation compensation. They restricted their 6-DOF master and 7-DOF slave to just 2-DOF, matching the two axes for each. For our system setup, the first kind of position-position command restricts the third joint to 0°.

For this teleoperation system, the slave's position dictation by the master is described by the following equation:

$$\boldsymbol{q} = K\boldsymbol{u} \tag{6.2}$$

Here, the vector $\boldsymbol{q} = [\theta_{s1}\ \theta_{s2}\ \theta_{s3}\ \theta_{s4}\ \theta_{s5}]^T$ contains the joint angles to be sent to the slave, $K = diag(k_1\ k_2\ 0\ k_4\ k_5)$ is matrix of scaling constants, and the input vector from the master controller is of the form [8]:

$$\boldsymbol{u} = \begin{bmatrix} \theta_{m1} \\ \theta_{m2} \\ 0 \\ \theta_{m3} - \theta_{m2} - \pi/2 \\ \theta_{m4} \end{bmatrix} \tag{6.3}$$

6.2 Position-Speed Command

This strategy entails using the position displacement of the master controller to command a velocity of the slave device. In essence, the master operates like a joystick, the further the current position from the initial position, the faster the slave end effector will move. In [4], the displacement of the master q_m was multiplied by a matrix of constant values to command the speed of the two-wheeled mobile robot:

$$\begin{bmatrix} V \\ \phi \end{bmatrix} = \begin{bmatrix} k_v & 0 \\ 0 & k_a \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \tag{6.4}$$

Here, $[q_1 \ q_2]^T$ is the vector of master angular displacement values, the differences between current and start positions for each joint, $K = diag(k_v \ k_a)$ is a matrix of constant speed terms, and $[V \ \phi]^T$ is the velocity vector, with speed V and direction ϕ. In [13], the displacement between the initial and current positions and rotations were used to command the speed of a 6-DOF Puma560 serial manipulator for assistance in activities of daily living. The computation of the slave speed differs from (6.4) in that control algorithms on the slave side computer running a real-time operating system read the master differential position and multiplied it by a constant velocity factor and the refresh clock rate of the system [13].

In our system [8], we employ the position-speed command strategy in a different way. First, for the serial manipulator on the slave side, position-speed command is used *only* on the base joint. This allows the manipulator to swing quickly to any point in its 360° range. Second, we do not compute the speed for which the base joint moves using (6.4).

Instead, we employ a smooth, bounded, and continuous function for speed. The reason is that using something similar to (6.4), one has to use an *if*-statement to check if the computed speed is higher than the allowable maximum. By using a function that will allow us to set the lower and upper bounds, along with being smooth and continuous, that issue can be avoided. To achieve this, a sigmoid-like function is used to compute the velocity. The profile of this function has the signature "S-curve" common to the sigmoid family of curves [8].

$$
\begin{aligned}
\dot{\theta}_{s0} &= f(\dot{\theta}_{s0_{max}}, \Delta\theta_{m0_{max}}, \Delta\theta_{m0}) \\
&= \dot{\theta}_{s0_{max}} \left[atan\left(\left| \frac{2\pi}{|\Delta\theta_{m0_{max}}|} |\Delta\theta_{m0}| \right|^{c} / b \right) / \left(\frac{\pi}{2}\right) \right] sgn(\Delta\theta_{m0})
\end{aligned} \tag{6.5}
$$

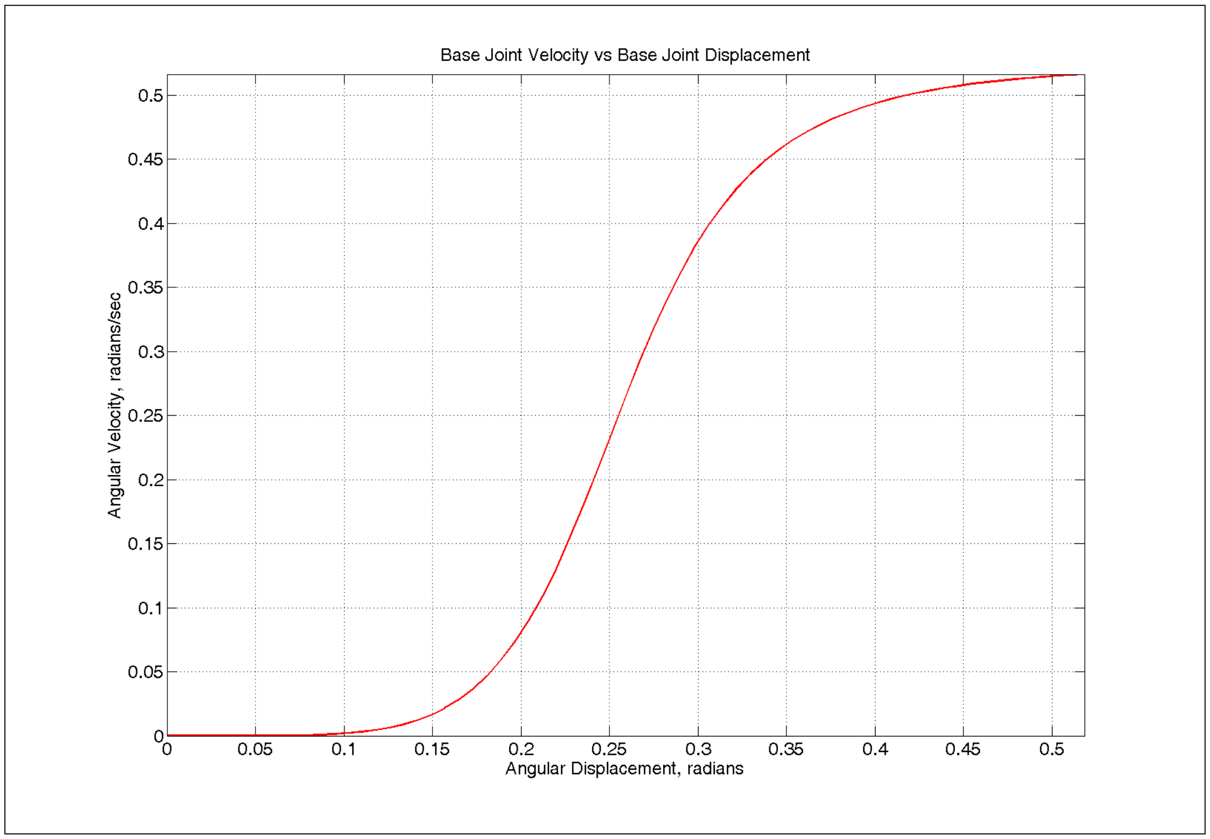


Figure 6.1: Plot for the speed function

Here, $\dot{\theta}_{s0,max}$ is the maximum angular velocity in $radians/s$, $\Delta\theta_{m0,max}$ is the amount of angular displacement between the start and current position required to achieve the maximum velocity in radians, and $\Delta\theta_{m0}$ is the current angular displacement also in the radians. For $\Delta\theta_{m0} = 0$, this function is zero, but for a small maximum displacement, the function does not reach its maximum velocity, but gets very close and is thus acceptable. Here,

$c = log(b)/log(a)$, where b and a are parameters that shape the curve for the function, with $b = 500.0$ and $a = 3.10$, respectively. The basis for (6.5) is found in [7].

In this mode the other joints are commanded the same as the previous section, but the command for the first joint as determined by (6.2) and (6.3) is ignored and instead comes from (6.5). This command strategy from this point forward will be referred to as Base-only Position-Speed Command, or BPSC.

7 INPUT SMOOTHING

The input values going from the master to the slave must be smoothed out to retard jerky motions that can cause the slave to become unstable and shut off. To achieve this, a low-pass filter is applied to the commanded joint angles prior to being sent to the slave device. The discrete form of the low-pass filter, also called the exponentially-weighted moving average, is of the form:

$$y_{n+1} = (1-\alpha)y_n + \alpha x \tag{7.1}$$

Here, α is a smoothing term with a value between 0 and 1. In the control program, the filter is coded in matrix form for each of the input joints of the master controller:

$$\boldsymbol{q}_{n+1} = A\boldsymbol{q}_n + B\hat{\boldsymbol{u}} \tag{7.2}$$

In (7.2), $\hat{\boldsymbol{u}}$ is equal to $\boldsymbol{q}$ from (6.2).

The matrices A and B contain the values necessary for the smoothing are are dependent upon the α values:

$$\begin{aligned} B &= diag(\alpha_1, \alpha_2, 0, \alpha_4, \alpha_5) \\ A &= diag(1-\alpha_1, 1-\alpha_2, 0, 1-\alpha_4, 1-\alpha_5) \end{aligned} \tag{7.3}$$

From (7.3), it can observed that for $\alpha_1, \alpha_2, \alpha_4, \alpha_5 < 0.5$, there is a strong bias towards the previous commanded joint values as opposed to the current input values. That property for this application is desired because it prevents large fluctuations in input values from shutting down the slave device and/or causing fast, jerky motions from sudden changes in values. In this system, $\alpha_1 = 0.02$, $\alpha_2 = 0.05$, $\alpha_4 = 0.05$, and $\alpha_5 = 0.05$. It was determined through trial and error that these values were best suited for the system. There is no α_3 because the slave robot's third joint is not in operation [8].

8 COLLISION AVOIDANCE

8.1 Avoiding the Table Top

An important issue for teleoperation setup was making certain that the master controller could not drive the slave to hit the table, consequently damaging the robot. This is achieved with using the input smoothing and the forward kinematics together. After a new set of joint angles are computed via (7.2), the new values are fed into the forward kinematics to obtain the corresponding position $\boldsymbol{x}$. The z_{n+1} is checked against the allowed minimum height z_{min}. If the computed z_{n+1} is less than z_{min}, then the computed joint angle values are ignored by the control program. Time delay in the system still allows the slave to possibly slip below z_{min}, but only slightly. In that situation, the control program will only allow the robot to move when a set of joint angles computed by (7.2) has a corresponding $z_{n+1} > z_{min}$.

8.2 Self-Collision Avoidance

The flexibility of serial manipulators allows them to reach locations in the workspace with relative ease. This is especially true for kinematically redundant serial manipulators. When controlling a serial robot, one should take heed of the particular environment of operation and employ some method of collision avoidance to ensure no part of the manipulator collides with objects in the workspace. However, with serial manipulators, there is just as big of a possibility that the robot will collide with itself. Some collisions can be avoidance via checks for joint limits, but for manipulators with many links and joints, this is not nearly good enough. Even before planning collision avoidance for the workspace, care should be taken to address the issue of self-collision avoidance.

Many approaches to self-collision revolving around representing the links and joints with volumes or employing protective volumes around the links. Seto et. al. [10], employed such a scheme for self-collision avoidance. Using what they called "Representation of Body by Elastic Elements" or RoBE, segments of each link were represented by a hard kernel surrounded by an elastic outer covering.

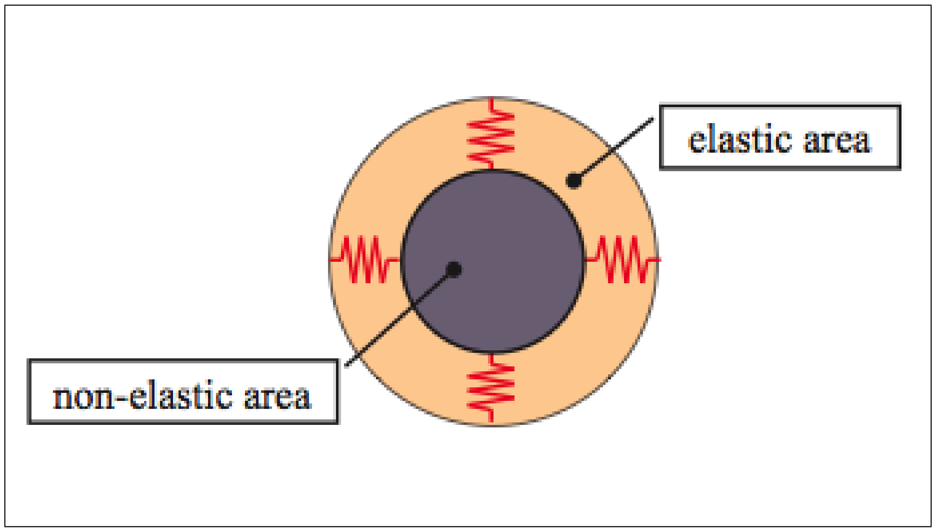


Figure 8.1: Elastic element[10]

The non-elastic area, the kernel, completely encompasses its section of a link, and the elastic layer on top of that. Here, the purpose of algorithm was to stop any movement that would cause the robot to collide with itself, while using the elastic layers of the elements of generate repulsive avoidance forces. These elements are either spheres or cylinders because of the ease of computation[10]. Each of these elements are located along the arms and body of the robot at chosen locations referred to as *control points*.

The Skeleton Algorithm's [9] authors used a slightly different approach. Modeling their humanoid robot as a kinematic skeleton, *collision points* are placed along the sections or the torso and arms of the mechanism. For each point, a sphere is created to surround that point, and each sphere is geometrically largely enough to cover the the physical dimensions of the segment it encompasses, plus an addition buffer for the collision avoidance [9].

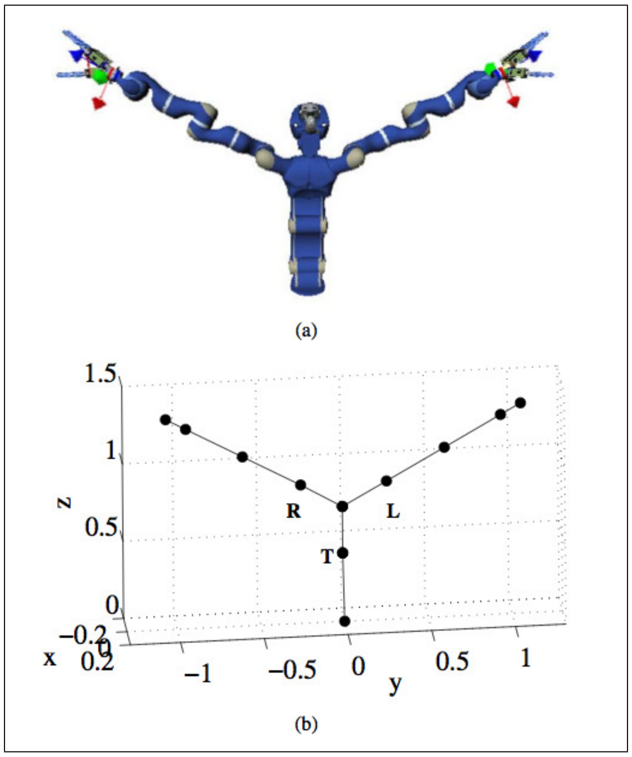


Figure 8.2: (a) DLR Justin, (b) kinematic skeleton [9]

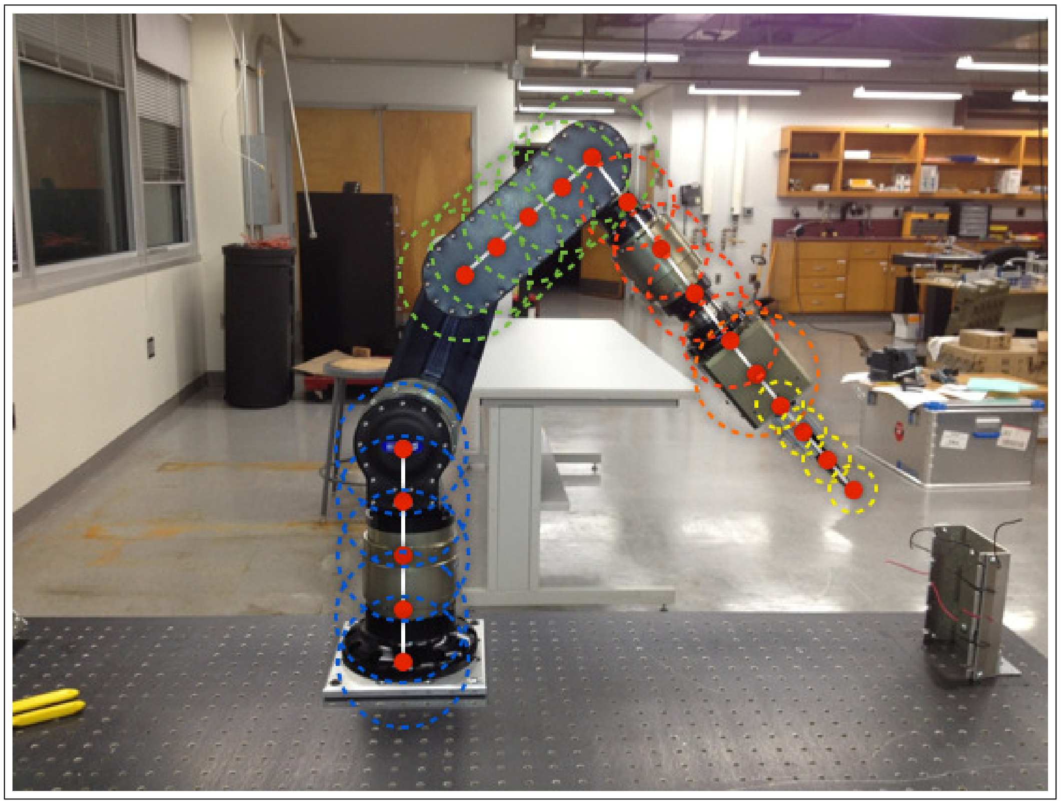


Figure 8.3: Collision control points and virtual circles

Using enough spheres, the volume surrounding any link approaches that of a cylinder, with two semicircles caping each end. In their application, the authors also used said method to generate repulsive forces to prevent the arms of the robot from colliding [9]. Since the collision points are located along the lengths of the links, the algorithm simply computes and checks each sphere by employing the forward kinematic equations. For the control points along each link, the distance of the collision control point from the previous kinematic origin is substituted into the transformation matrix between two subsequent frames [9].

For the application in this paper, the 5-DOF slave robot employs the scheme describe above similarly for self-collision avoidance. Let l_3 be the length separating each control point along the link a_3, which is equal

$$l_3 = \frac{\|a_3\|}{s_n} \tag{8.1}$$

where s_n, an integer, is the number of segments in which to divide the link. Also, let $r_{B,i}$ be the radius of the *ith* sphere along the length of the base link, and $r_{3,i}$ to be the radius of the *ith* sphere along the length of link a_3, for $i = 1, ..., s_n + 1$.

When the distance between a collision point along the base link and one along the third link a_3 is less than $\|r_{B,i} - r_{3,i}\|$; that is, when one circle on the base touches or intersects a circle on a_3, the function returns a code corresponding to a possible collision of the base link and the third link. Using this code, a program controlling the slave device can halt the movement of the slave in its workspace thus avoiding a possible self-collision.

For the fourth link, a slight modification was employed to deal with the very non-uniformity of the link. The fourth link was divided into three sections in the direction of x_4. The first section runs from O_4 (Figure 4.2) to the first wall of the end effector tool box. The second section runs the length of the end effector tool box, from the first wall to the second wall along the direction of x_4. The final section runs from the second wall of the tool box to the tips of the gripper fingers. With the exception of the tool box having only one circle to encompass its whole area, the the approach for possible collision detection between the base link d_1 and the last link a_4 is the same as previously described above for link a_3.

9 EXPERIMENT AND RESULTS

9.1 Setup

For each experiment, the user will be asked to move the master controller and correspondingly move the slave robot to cut one wire at each wire station in the slave robot's workspace. The data taken from each user in the experiment is time, ease-of-use, and the number of wires that each user cut. Time is measured in seconds and is tracked by the control program. The ease-of-use metric is scale from 1 to 5, with 1 being "too hard" and 5 being "very easy".

Volunteers for the experiment were recruited at random, and were FAMU-FSU College of Engineering students. The work of Alise in [2] also entailed the use of human subjects. The author had participants trace various shapes in a task-tracking experiment. The participants in the study were received training prior to the experiment, and the experiment was only given when they felt comfortable and confident. Unlike the [2], the human subjects in this experiment were not given training but rather a brief demonstration by the experiment facilitator after being introduced to the system. They were asked after the demonstration about their understanding of the operation of the system and what they were to accomplish. After it was clear each participant understood all the aspects of the experiment required of them, the experiment was then administered.

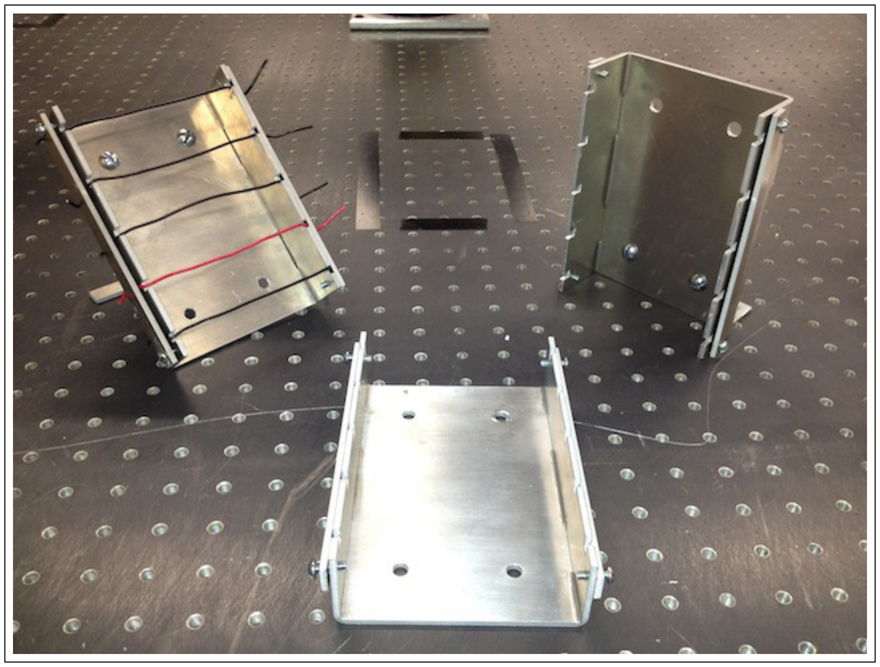


Figure 9.1: Wire stations for holding wires in the experiment

For each experiment, the user will press and hold the white button on the Phantom Omni's stylus to move the Schunk robot in its workspace. The user will be required to cut one wire at each of the three wire stations. The first station is flush with the horizon of the table. The second station is at a 45° angle with the horizontal. The third station is vertical with respect to the horizontal. At each station, the user will cut the red wire, the other wires the user should not cut will be black. When the user needs to cut a wire, the user will press and release the blue button on the Omni's stylus, which will cause the wire-cutting apparatus on the slave robot to quickly close and then reopen. If the user finds it difficult to place the manipulator and cut a wire at a particular station, they are allowed to move on to the next station, however by doing so they forfeit the chance to engage that station and cannot go back. For the angled and vertical stations, if the user knocks them over, they also lose the opportunity to engage that station.

9.2 Results

Table 9.1: Raw data from both experiments

Participant	PPC time (sec)	BPSC time (sec)	PPC score	BPSC score
1	83.5	115.5	2	1
2	97.0	64.5	3	4
3	76.5	92.75	4	2
4	173.5	73.0	2	3
5	107.5	125.25	3	2
6	90.5	230.0	5	3
7	65.0	77.75	3	5
8	81.5	43.75	2	3
9	196.0	394.5	3	2
10	103.75	447.25	4	2

Table 9.2: Time quantities for the experiment.

Quantity	PPC (sec)	BPSC (sec)
Average	107.4750	166.4250
Standard Deviation	42.9739	144.0173

Table 9.3: Score quantities for the experiment.

Quantity	PPC	BPSC
Average	3.1	2.7
Standard Deviation	0.9944	1.595
Mode	3	2

Table 9.4: Average time and score confidence intervals for PPC

α	time (sec)	score
90.0%	$88.72 < \mu < 126.23$	$2.67 < \mu < 3.53$

Table 9.5: Average time and score confidence intervals for BPSC

α	time (sec)	score
90.0%	$103.58 < \mu < 229.27$	$2.19 < \mu < 3.21$

The confidence interval addresses the problem of how well a computed statistic reflects the value of the population. It places an interval on a range of values where the true value in question is likely to reside for subsequent experiments. This value is subject to a chosen confidence level. Standard confidence levels used are 99.0%, 95.5%, 90.0%[3]. For sample sizes $n < 20$, the confidence interval is given by:

$$[\bar{x} - t(\alpha)S_{\bar{x}}] < \mu < [\bar{x} + t(\alpha)S_{\bar{x}}] \tag{9.1}$$

Here, $\bar{x}$ is the computed mean, $t(\alpha)$ is *Student's t* statistic, where α is the confidence level (in decimal). Standard values of $t(\alpha)$ is are easily researched in a standard computed table given α and the number of *degrees of freedom*, defined as the sample size n minus one, $d = n - 1$. The parameter μ is true mean. The standard error $S_{\bar{x}}$ is defined as:

$$S_{\bar{x}} = \frac{\sigma}{\sqrt{n}} \tag{9.2}$$

Here, n is the sample size, and σ is the standard deviation[3].

The average scores for the PPC and BPSC indicate users thought the PPC strategy was easiest to use for the given task. Only four of the the ten participants gave the BPSC strategy a higher score than the PPC strategy. It is interesting to note that these scores came from those who took less time to cut the wires with BPSC than PPC. One of those four users did give BPSC a higher score despite taking longer with that command strategy. The most frequently given scores for the PPC and BPSC strategies respectively were 3 and 2. The PPC strategy received more higher end scores (4's and one 5) than BPSC which only received two scores from that part of the scoring spectrum. A very plausible reason for PPC receiving a higher score is because of the very small amount of ambiguity in the relationship between where the end point of the input device is and the location of the end-effector of the remote arm is in their respective workspaces. Although each participant had to start with PPC strategy run first, it is believed that the dominating factor in the user's ability to perform the given task was the spatial relationship between where the master controller's end point is in its workspace and where the remote arm is in its workspace. In order to reach the extremes of the remote arm's workspace using PPC strategy, one must move the master controller to the extremes of its workspace. The fact that only six of the ten participants gave PPC strategy a higher score shows there is at most a weak correlation between the given scores and which command strategy was tested first. The order of the experiment had very little effect on the outcome of the experiment. There is at least 90.0% confidence that the if experiments were conducted again with another ten random participants, the range of $2.67 < \mu < 3.53$ would contain the true population mean. If the experiments for BPSC were conducted again with other 10-sample populations, it is believed that mean score for 90.0% of the new cases would fall in the range of $2.19 < \mu < 3.21$.

From the above Table 2, the average time PPC is much lower than that for BPSC. In fact it is shown that only three of the ten participants took more time to cut the wires with PPC than BPSC. In addition, many users took much longer with the BPSC strategy. The last participant from table 1 took 103.75 seconds (1 minute 73 seconds) with the PPC strategy while taking 447.25 seconds (7 minutes 45 seconds) for BPSC while only being able to cut 2 wires. The users were not giving any training prior to the experiment as in [2], only a demonstration given by the facilitator. Actual training could have improved times so that they would have been more familiar with the feel of the operation. Even with training, it would be expected that PPC strategy would fair better in times and scores than BPSC. It could be argued that the average for PPC strategy is high because one participant from took particularly long for the experiment, and was only able cut one wire (knocking the other holders off the table). Likewise, participant 10's time for BPSC also pulls up the average time for BPSC significantly. A confidence level of 90.0% suggests that the average of the BPSC for reruns of the experiments with other random populations would fall between $103.58 < \mu < 229.27$ seconds for 90% of the new cases.

10 CONCLUSIONS

It is clear that the PPC strategy is better for the type of application where the users can immediately see the remote arm in its workspace. More plausibly, this command strategy is best suited for situations where the user, wither explicitly or implicitly, expects that the position of the remote arm's end-effector is a strongly related to the positioning of the master controller's end-effector. The implementation of PPC strategy presents little ambiguity between the position of the master controller in its workspace and that of the end-effector of the remote arm. When the user moves the master controller to one extreme or the other, so does the remote arm also move to its extremes. However, a drawback of PPC strategy, as also discussed in [4], is that the scaling between the master controller positions and that of the remote arm can be very sensitive. A small change in position on the user side could create a large and undesirable change in position in the remote arm. Possibly, the BPSC is better suited for situations where the user needs to access all 360° of the workspace, and the master controller is much more limited than the remote arm, and there needs to be a strong buffer against the sensitivity of the master controller. A possible scenario where BPSC strategy would excel would be a teleoperation setup where the user was presented with two views, one of which comes from a camera mounted on the serial manipulator looking directly in front of its task or target. The other view would be orthogonal to the the remote arm's end-effector and come from a camera suspended in its workspace. Alternatively, the second orthogonal view could be supplied by an aerial vehicle solely dedicated to tracking the manipulator in the workspace. Since the user would mostly like have less of an expectation that the positioning of the master was related in scaled or absolute terms to that of the remote arm, and with less sensitivity about the base using (6.5), BSPC would be better here as opposed to PPC.

11 FUTURE WORK AND CONSIDERATIONS

The serial manipulator used in this work will be integrated with an unmanned ground vehicle to serve as the primary tool for manipulating the environment. The methods presented here are possible candidates for operation of the manipulator. Special consideration should be taken as to where the manipulator should be mounted as it vastly effects the arm's workspace. For operation onboard a mobile base, collision with the mobile base as well as collision with passing objects should be considered. Developing a method to control all five degrees of freedom of the remote arm that is not cumbersome is in the works.

APPENDIX 1: Getting Started with the Schunk 5DOF

What you will need

In order to program and/or control the Schunk 5-DOF serial manipulator, there are several pieces of software that must first be installed. These are all linked and not having one install will prevent a user from being able to adequately do research and/or control with the manipulator. Also, diagnosing problems and clearing errors is also not feasible with many of these pieces of software. They are, in no particular order:

- CAN SDK
- CANScope
- PowerCube software and SDK
- Visual C++ 2008 Express Edition

Installing the CAN Software Development Kit (SDK) is essential as the PowerCube Software and Application Programming Interface (API) depends on the CAN protocol under the hood for communicating with the manipulator. The CANScope allows for one to see the data being sent to the robot as it is happening or after the robot is finished moving. Visual C++ 2008 Express Edition is an Intergrated Development Environment (IDE) for writing C and/or C++ programs. The PowerCube API is written in C, but can be used in C++ programs. More details about these softwares and installation in subsequent sections.

Contacting the manufacturer

If for some reason there is a question not answered in this guide or the Schunk documentation, you should contact the company's U.S. representatives.

Schunk Intec, Inc

211 Kitty Hawk Drive Morrisville, NC 27560

Tel. +1-919-572-2705

Fax +1-919-572-2818

info@us.schunk.com

www.us.schunk.com

About this guide

This guide is not intended to, in material form or in spirit, replace, supplant, circumvent or extend any documentation provided by the manufacturer of the serial manipulator. It is meant to help the next user of the device get off to a better and smoother start than the author of this guide by walking him or her through the steps taken by the author to bring everything together to get the robot working and deal with errors and debugging issues. This guide is also not intended to be a primer on C or C++ programming, it is assumed the reader has a fair grasp on said programming languages.

Installing CAN SDK and CANScope

First, one needs to locate the disc for the CAN SDK that comes with the serial manipulator. If the disc cannot be located, please contact Schunk to obtain either a downloadable version or a disc that can be mailed to your location.

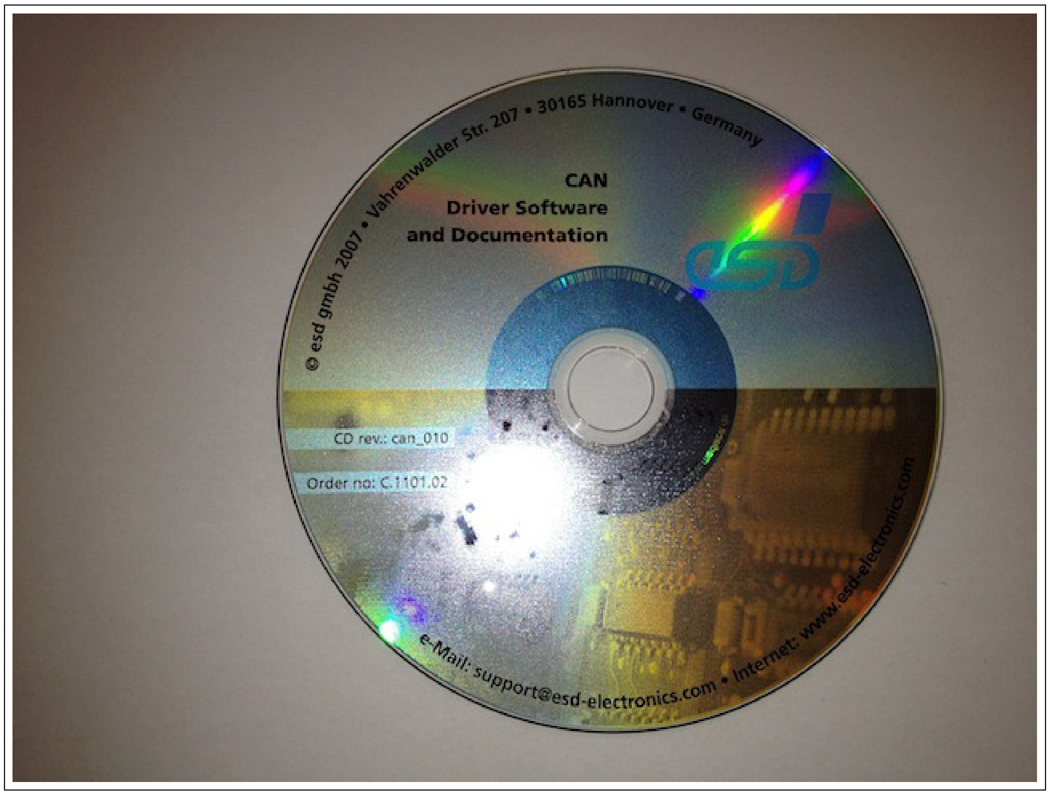

© esd gmbh 2007 • Vahrenwalder Str. 207 • 30165 Hannover • Germany
CAN
Driver Software
and Documentation
esd
CD rev.: can_010
Order no: C.1101.02
e-Mail: support@esd-electronics.com • Internet: www.esd-electronics.com

Figure 11.1: This CD-ROM contains the CAN SDK and installer

Make sure you have a copy the CANScope software installed somewhere on your computer. If you do not already have this software, please contact Schunk, as this software DOES NOT come on the CAN SDK disc. They will most likely email the software in a .zip file.

Instructions for installing the software:

- Place the disc into the CD tray.
- Use Windows Explorer to view the contents of the CD-ROM.

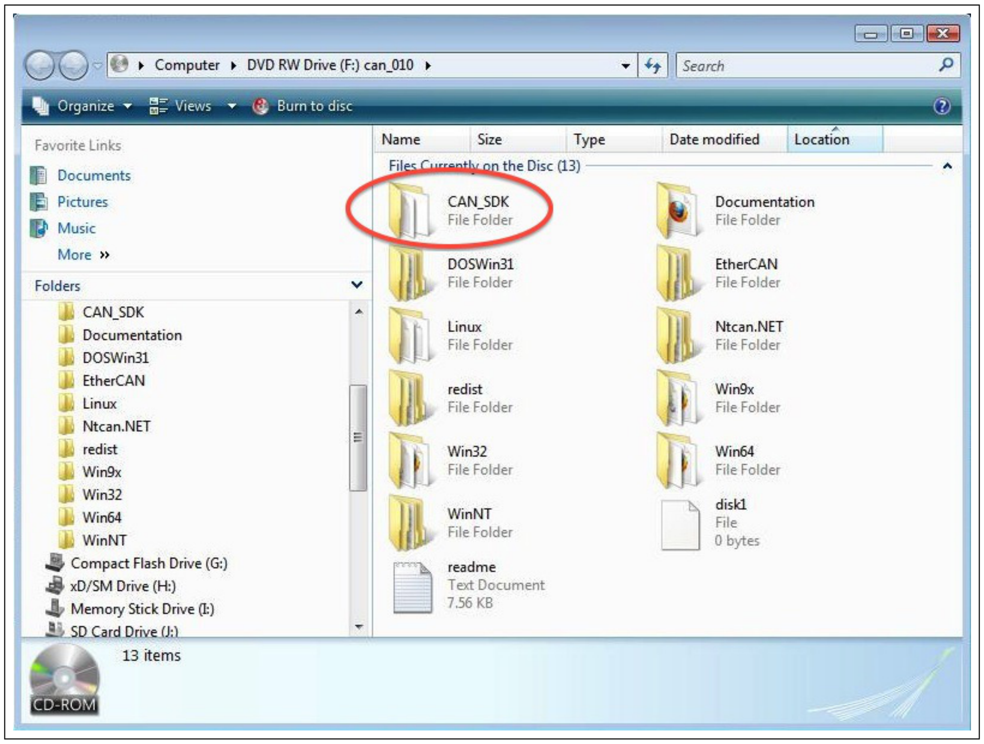


Figure 11.2: View of disc contents

- Double-click the “CAN_SDK” folder to reveal its contents.

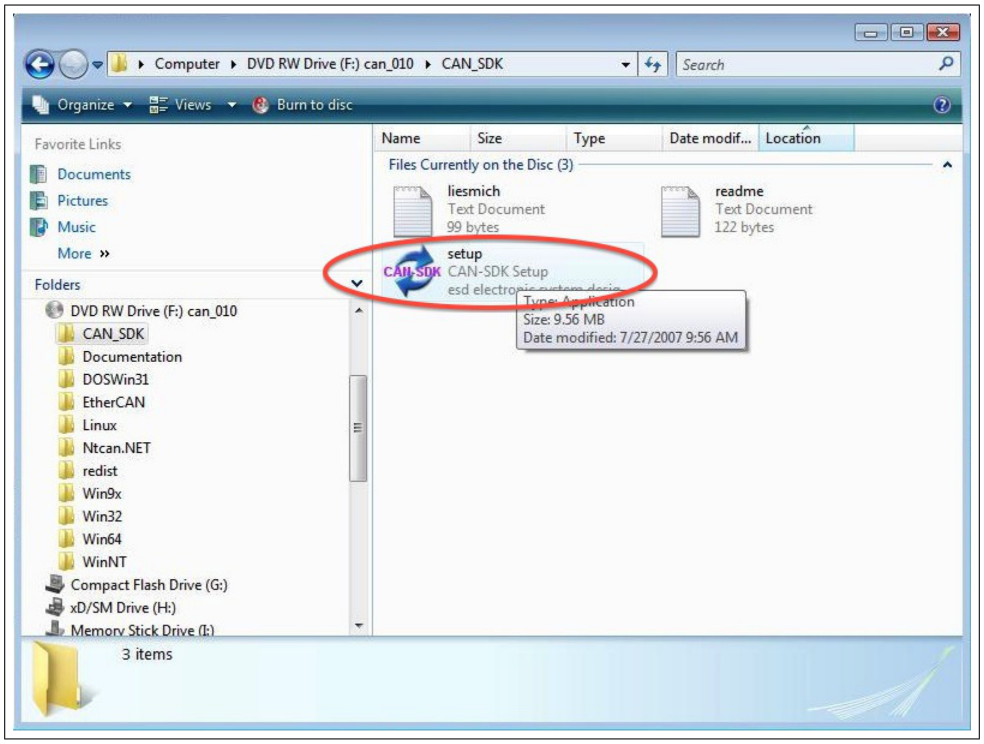


Figure 11.3: CAN SDK installer application

- Double-click the "CAN-SDK Setup" installer application, and follow the instructions.

- Navigate to the folder where the CANScope installer application is stored

- Double-click the installer application and follow its instructions.

OPTIONAL: The following instructions are optional, but could help tidy things up and keep things consistent. The CANScope software is separate from the CAN SDK and associated tools it comes with, so it will install in a separate location. However, one could put the CANScope software inside of the folder with the other CAN tools, as CANScope is a CAN tool itself. If you desire to do this, do the following:

- Find the folder that contains CANScope and all associated files.

- Right-click and choose "cut".

- Navigate to the folder containing the CAN SDK and all associated files, this folder is labeled “CAN_009”.
- Insider the “CAN_009” folder, find the “CANtools” tools.
- Paste the “CANScope” folder and associated files insider of the “CANtools” folder.
- The CANScope software and associated files will now be located in the same folder as the rest of the CAN tools.

Installing the PowerCube Software and SDK

Make sure you have located and secured the CD-ROM containing the PowerCube software. This software includes the PowerCube test suite for testing, debugging, and clearing errors for each module of the serial manipulator. If you do not have this CD-ROM, contact Schunk for a copy of the software via download or mail.

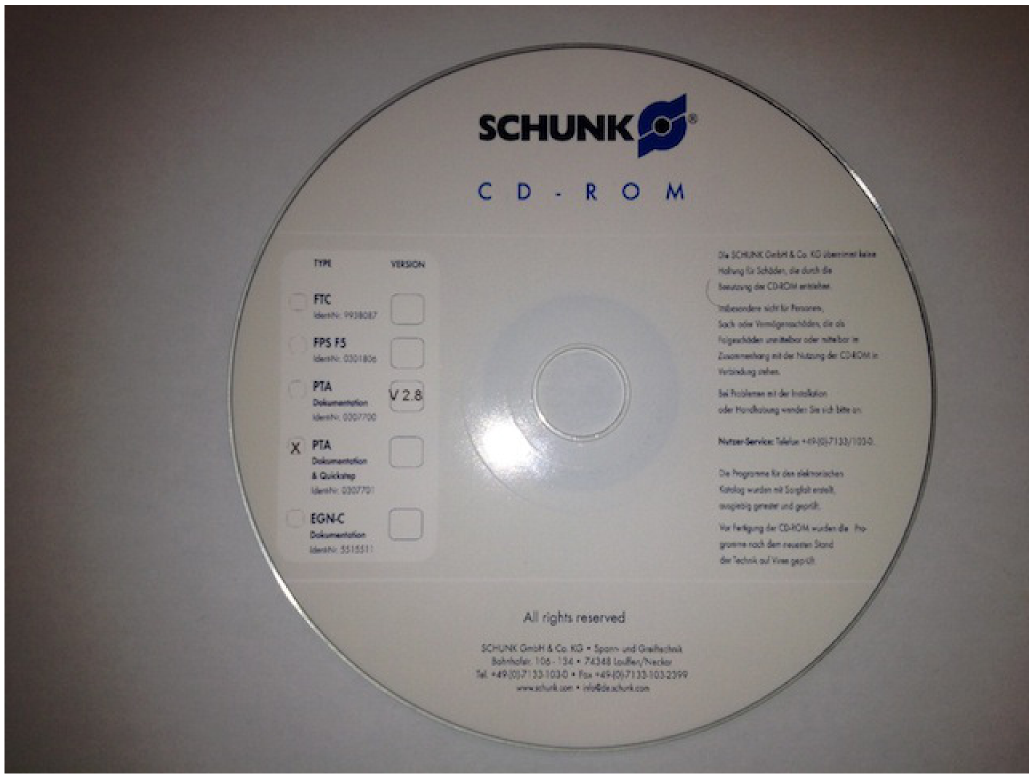


Figure 11.4: This CD-ROM contains the PowerCube software

Install the software by doing the following:

- Place the CD-ROM into the CD tray.
- Use Windows Explorer to view the contents of the CD-ROM, your current folder location should say “SCHUNK”.

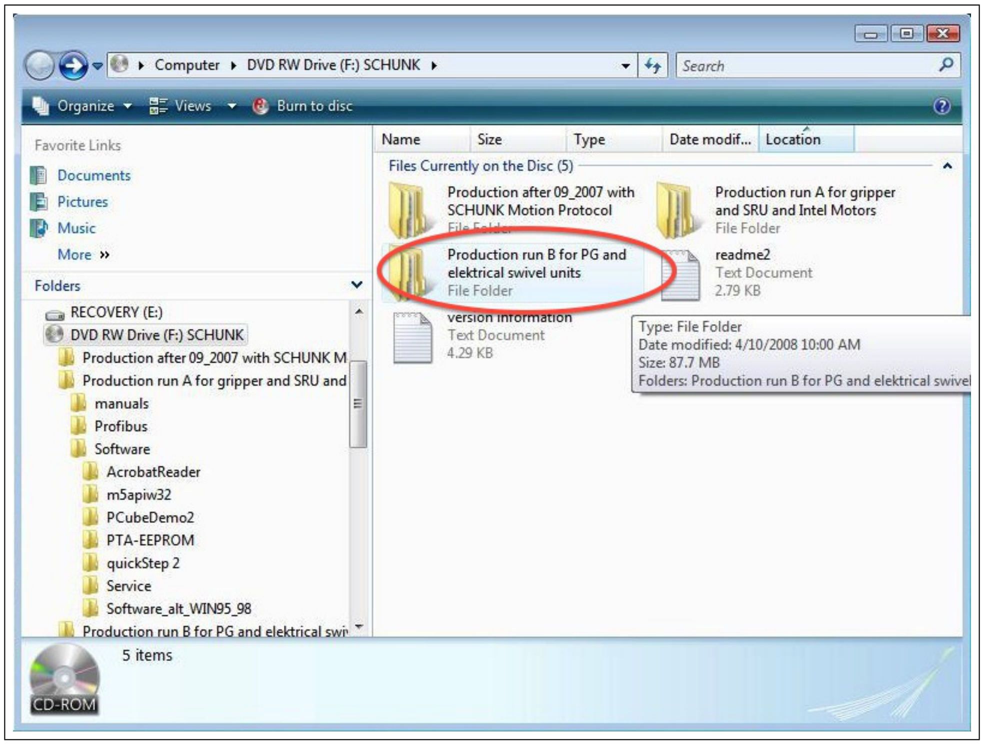

Computer ▸ DVD RW Drive (F:) SCHUNK ▸
Search
Organize
Views
Burn to disc
Favorite Links
Documents
Pictures
Music
More »
Folders
RECOVERY (E:)
DVD RW Drive (F:) SCHUNK
Production after 09_2007 with SCHUNK M
Production run A for gripper and SRU and
manuals
Profibus
Software
AcrobatReader
m5apiw32
PCubeDemo2
PTA-EEPROM
quickStep 2
Service
Software_alt_WIN95_98
Production run B for PG and elektrical swi
Name
Size
Type
Date modif...
Location
Files Currently on the Disc (5)
Production after 09_2007 with SCHUNK Motion Protocol
File Folder
Production run A for gripper and SRU and Intel Motors
File Folder
Production run B for PG and elektrical swivel units
File Folder
readme2
Text Document
2.79 KB
version information
Text Document
4.29 KB
Type: File Folder
Date modified: 4/10/2008 10:00 AM
Size: 87.7 MB
Folders: Production run B for PG and elektrical swivel
5 items
CD-ROM

Figure 11.5: Contents of CD-ROM when first viewed

- Navigate to the folder containing the installer file: “SCHUNK” ➾ “Production run B...” ➾ “Software” ➾ “PowerCube Software”.
- You should now see the PowerCube software installer file.

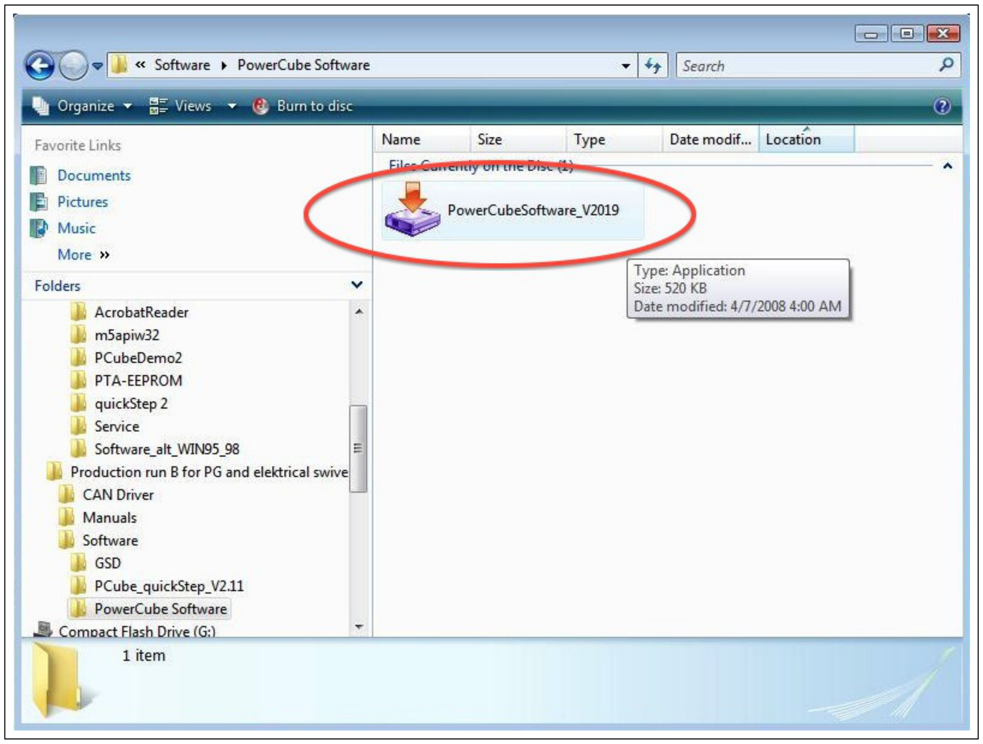


Figure 11.6: The folder containing the installer file

- Double-click the installer file and follow any instructions it gives.
- The software will be installed on the computer in a folder labeled “amtec robotics”.

Installing Visual C++ 2008 Express Edition

Visual C++ 2008 Express Edition is an Integrated Development Environment (IDE) for writing, compiling, debugging, and executing software programs written in C, C++, or and mixture of both. Please make sure you have located and secured a copy of the installer for this program. If you do not have it, do one of the following:

- Consult the professor in charge of the lab to see if a copy of the executable file for VC++ 2008 Express “vcsetup.exe” is saved on a secure external hard drive or some other location. Obtain the file and put it on the **C:** drive on your computer.

- If the above is not possible, the only other option is to download it from the Microsoft website. As of March 6, 2012, the direct link to the download page for the software is `http://www.microsoft.com/download/en/details.aspx?id=6506`, or type “visual studio 2008 express” into a search engine and obtain it either from Microsoft’s own site or another safe site. Make sure the executable file says “vcsetup.exe”.

- The setup for Visual C++ 2008 Express may prompt you to install other supplementary components such as the .NET Framework. If prompted, you will need to install these items as they are necessary for the VC++ program.

Hardware Setup

Now with associated software in place, we now will transition to setting up the hardware. To get started, make sure you have located and secured the following hardware items:

- Serial manipulator with PowerCube terminal attached.

- 24-volt, 21 - 27 Amp. constant power supply.

- Two CAN-USB converter modules.

- Two USB-A-to-USB-B cables

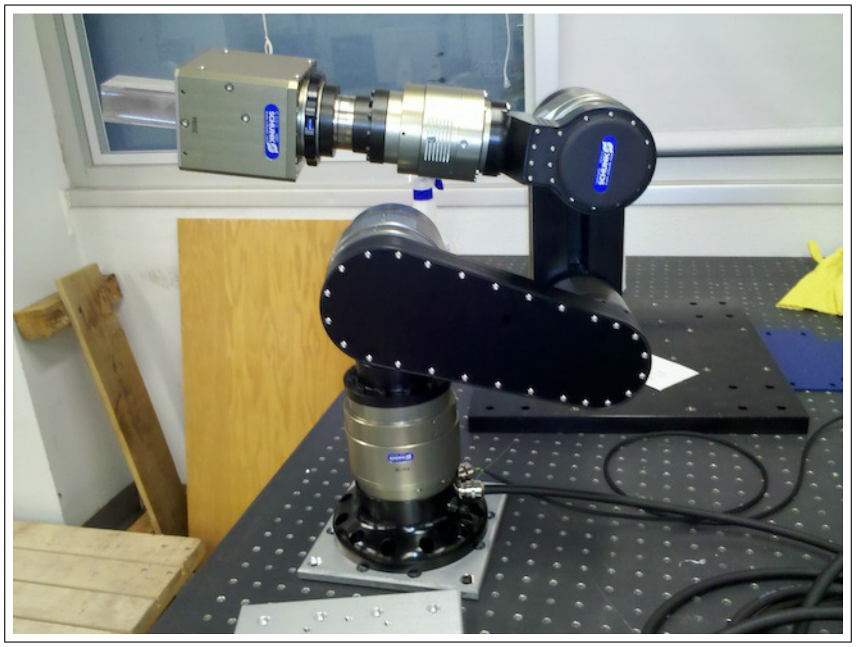


Figure 11.7: The 5-DOF serial manipulator

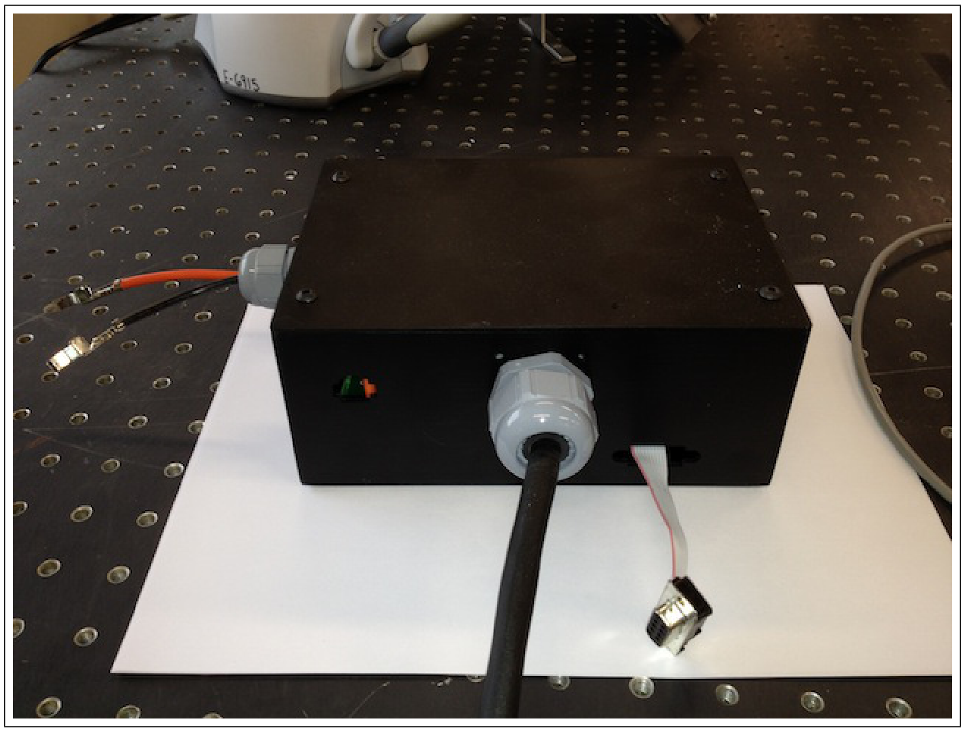


Figure 11.8: The PowerCube terminal

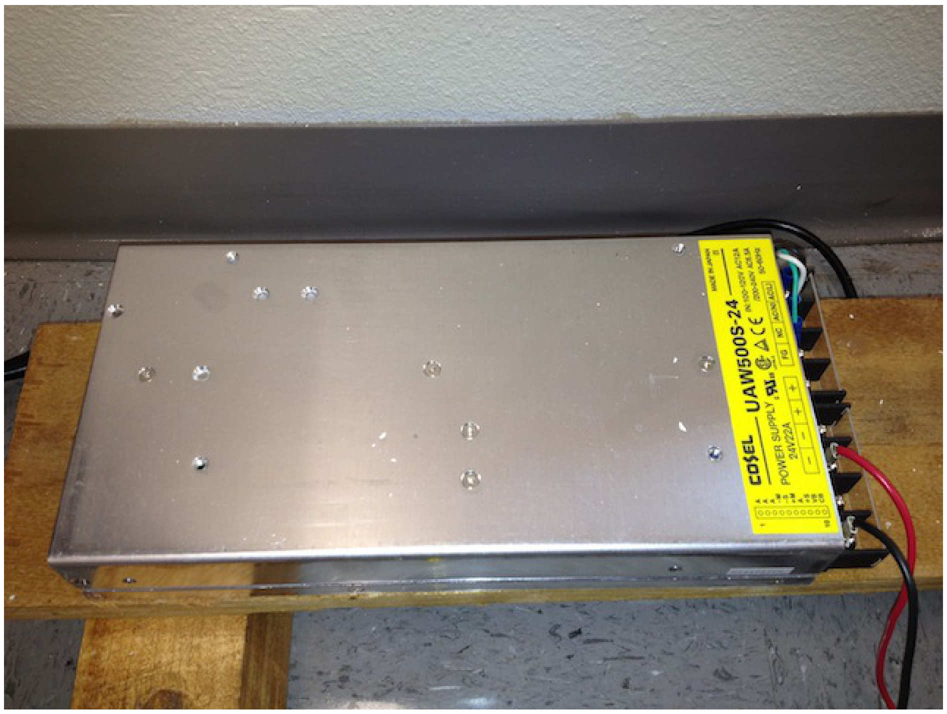


Figure 11.9: Constant voltage constant current power supply

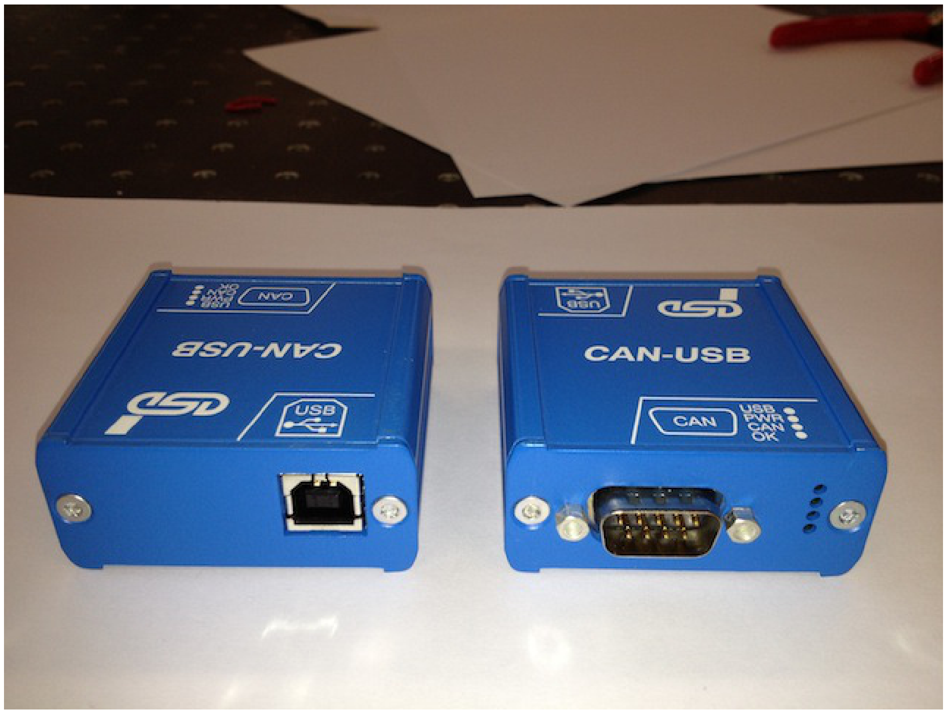


Figure 11.10: CAN USB modules

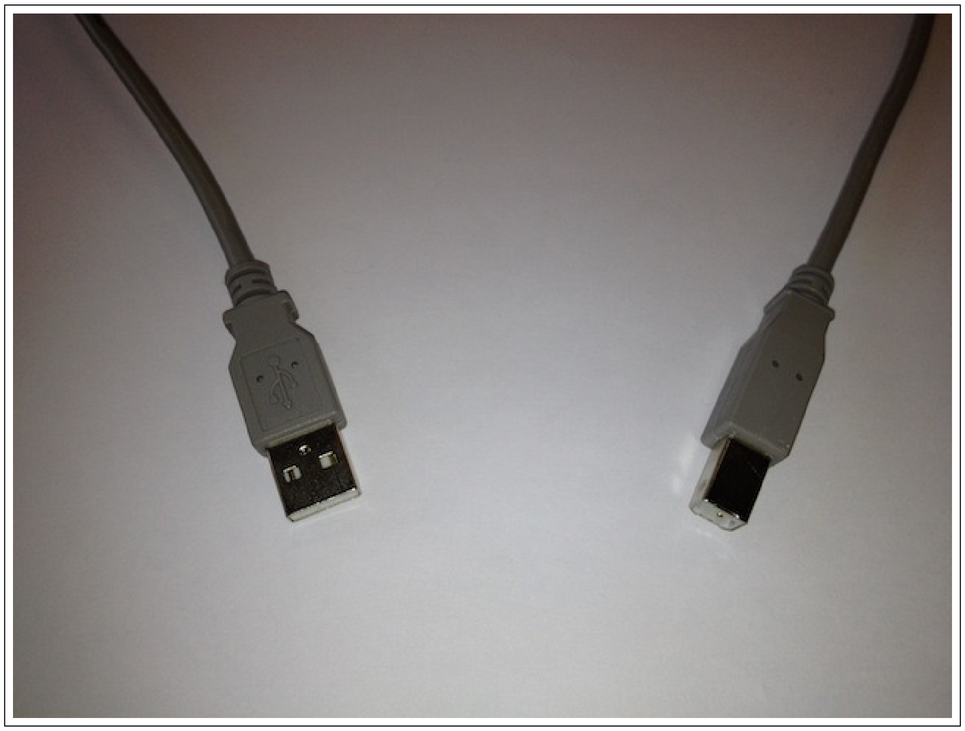


Figure 11.11: USB cable type needed for the setup

With the hardware ready, do the following:

- Secure the serial manipulator to a hard surface. There are holes drilled around the base of the manipulator where screws can be inserted and fastened. If using a metric table with pre-drilled holes for screws separated by a standard length (like 1-inch separation between holes left-right-up-down), then using a converter plate may be necessary. If that is the case, simply fasten the plate to the table, and then the manipulator's base to the plate.

- Now with the manipulator secured, locate the part of the base where two cords of different thicknesses feed into the robot. At the end of the thinner cord, there is a connector for the CAN USB module:

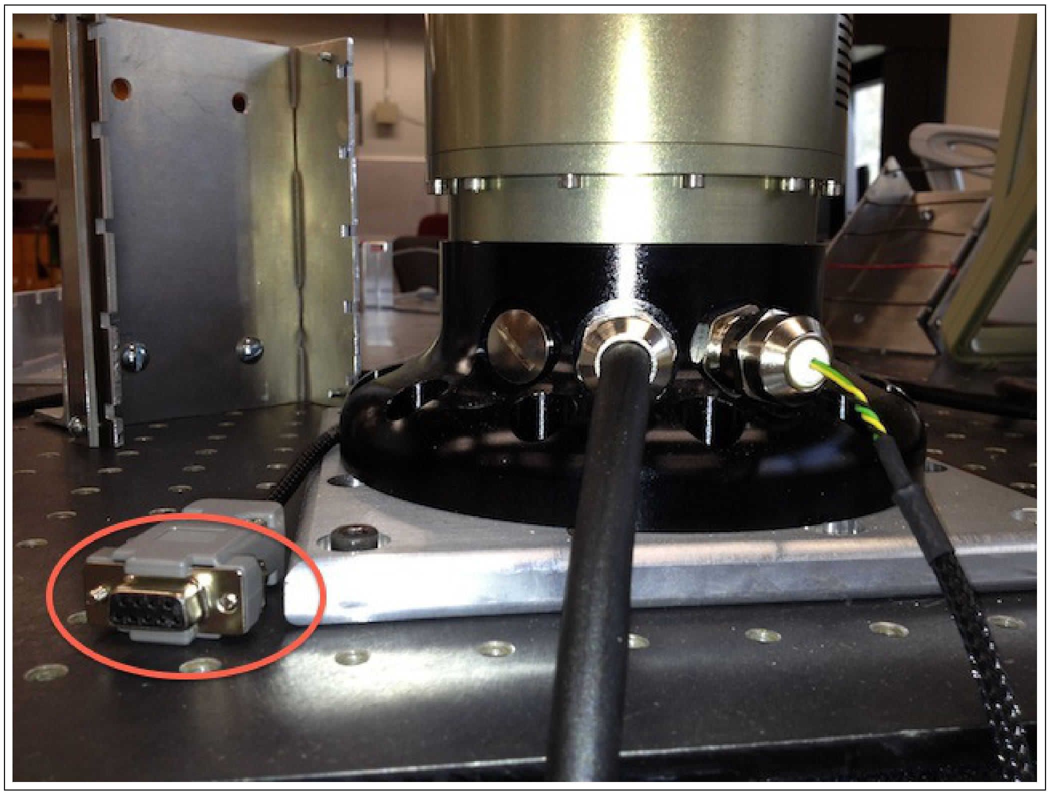


Figure 11.12: The cords with connection

- Take one of the CAN USB blue modules, and using the schematic on the module, connect and secure it to the connector indicated in the figure above.
- Take one of the USB-A-to-USB-B cables and connect the USB-B end to the CAN USB module from the previous step. Connect the USB-A end to a USB port on your computer. Data associated with robot such as position, velocity, module states, etc, will be fed into your computer via this connection. The robot comes with two cables that are approximately 3ft in length, but one may find that longer cables are needed. If ordering a longer cable, be sure that the connecting ends of the cable are the same as shown in Figure 11.11.
- The PowerCube terminal has a connection type similar to the one circled in red in

Figure 11.12. Take the second CAN USB blue module, and using the schematic on the module, connect and secure it to the connector on the PowerCube terminal.

- Using the second USB-A-to-USB-B cable, plug the USB-B end into the CAN USB module from the previous step, the place the USB-A end into a USB port on your computer.

- Finally, connect you power supply to the PowerCube terminal. There are two wires protruding from the black box (Figure 11.8), one for positive voltage (red), the other for negative voltage (black). Connect the positive wire from the terminal to the positive of the power supply, likewise the negative from the terminal should be connected to the negative of the power supply. Again, this power supply needs to be constant 24-volts, and constant current, with the acceptable current values between 21 - 27 Amperes.

Now, use the PowerCube software to detect the robot and view each individual module. Do the following:

- On your computer, locate a folder labeled "amtec robotics".

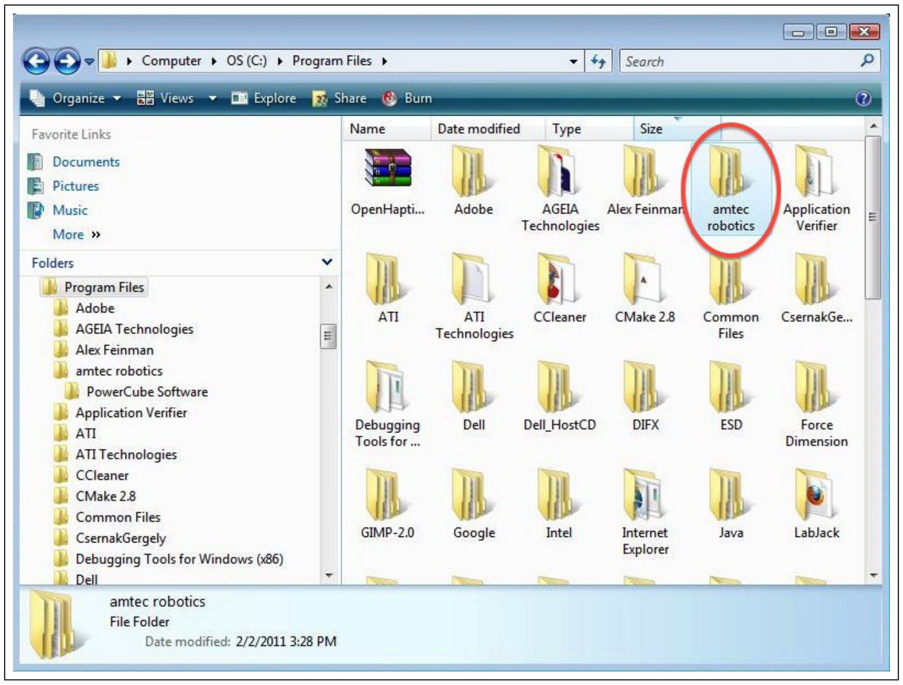


Figure 11.13: The “amtec robotics” folder containing all the PowerCube software

- Inside the “amtec robotics”, find the PowerCube software icon.

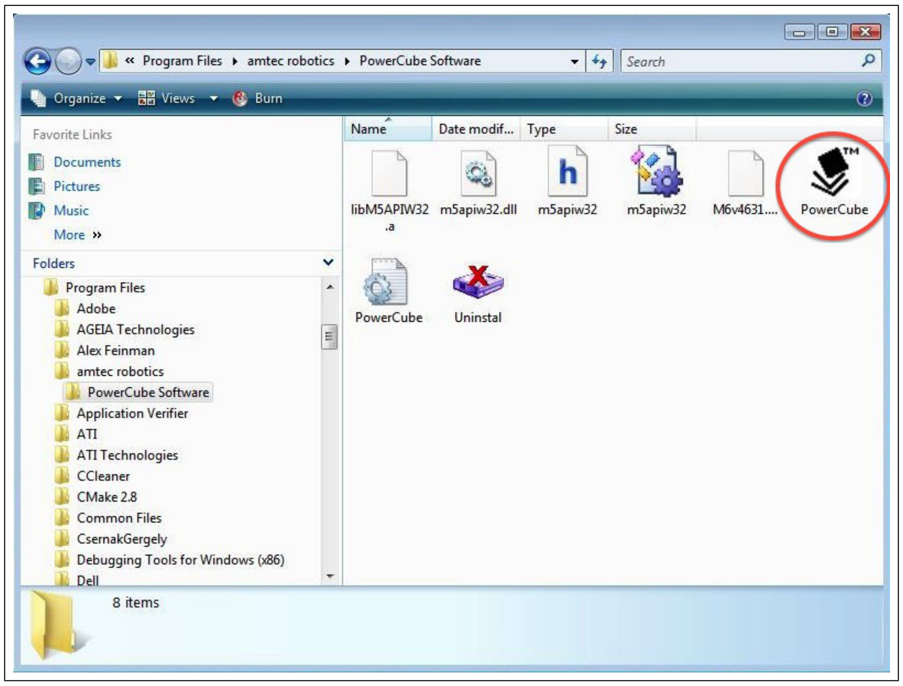
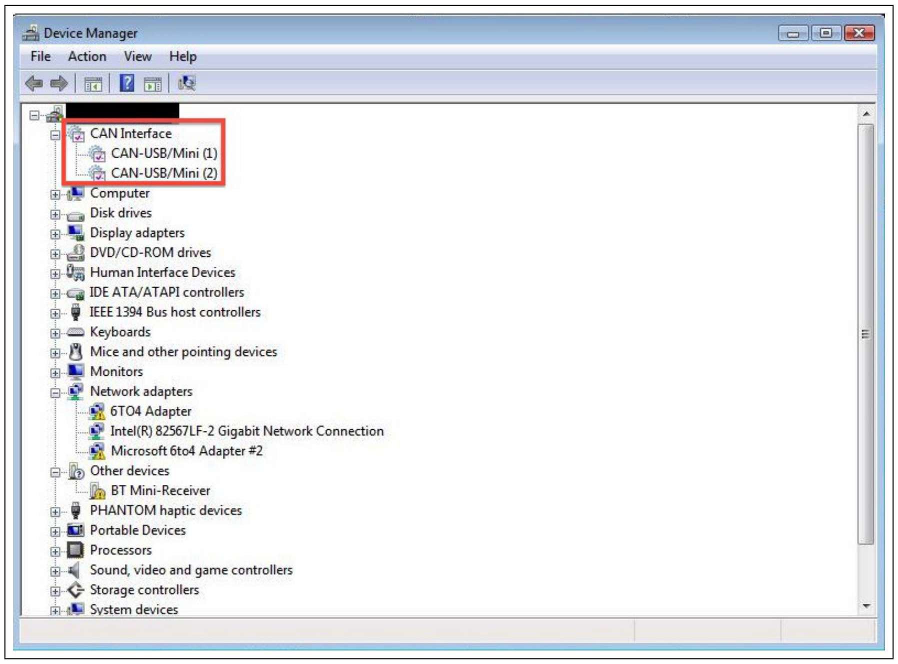


Figure 11.14: Contents of the “PowerCube Software” folder

Double-click the PowerCube software icon to launch the PowerCube software. When the software first launches, the white area of the window will be blank. To see if the software can detect the robot and all of its modules, click the button labeled "Scan". If the module numbers and information appear in the white window, then you can proceed and press the "Test" button. If nothing appears, do the following:

- First, use Windows Device Manager to determine where the robot's USB cables are connected. Click the (+) sign next to "CAN interface" in Device Manager, and for each listing underneath, right-click and select "Properties". For example, under the general tab in Properties, the location may read "Port_#0002", this means that one of the USB cables for the robot is connected to port 2. Record that number.

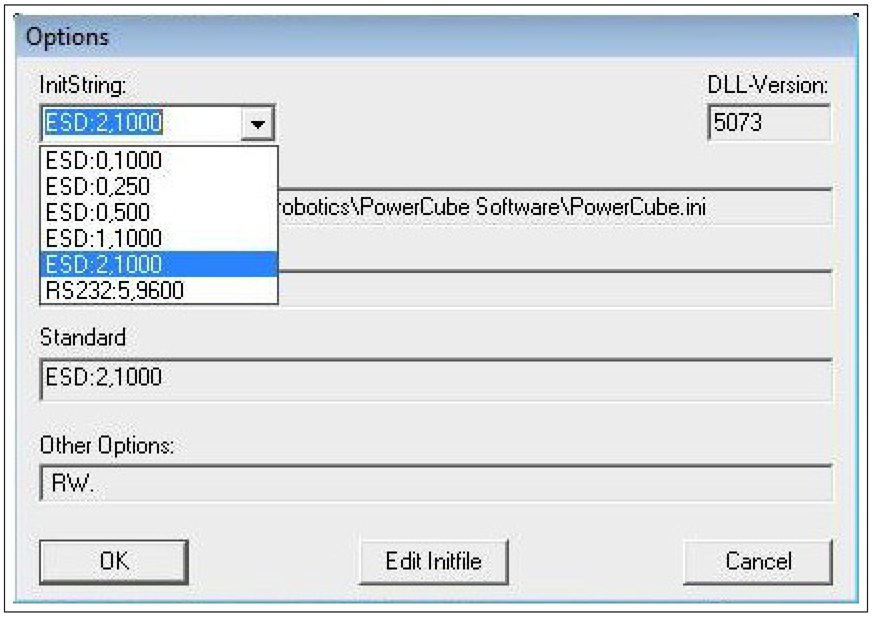


[redacted]
- CAN Interface
  - CAN-USB/Mini (1)
  - CAN-USB/Mini (2)
- Computer
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- Human Interface Devices
- IDE ATA/ATAPI controllers
- IEEE 1394 Bus host controllers
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
  - 6TO4 Adapter
  - Intel(R) 82567LF-2 Gigabit Network Connection
  - Microsoft 6to4 Adapter #2
- Other devices
  - BT Mini-Receiver
- PHANTOM haptic devices
- Portable Devices
- Processors
- Sound, video and game controllers
- Storage controllers
- System devices

Figure 11.15: CAN interface in Windows Device Manager

- In the PowerCube software, Click "Options".

- Under the "InitString" drop-down menu, make sure the current option reads "ESD:2,1000", where the middle number depends on your computer's ports. The options in this menu are communication specifiers used by the PowerCube Software and the SDK. For example, you may see a string "ESD:2,1000", where the "2" represents the port number. Change the current port number to one of the numbers you previously recorded from the Device Manager. Select "Ok".



InitString: ESD:2,1000

ESD:0,1000
ESD:0,250
ESD:0,500
ESD:1,1000
ESD:2,1000
RS232:5,9600

DLL-Version: 5073

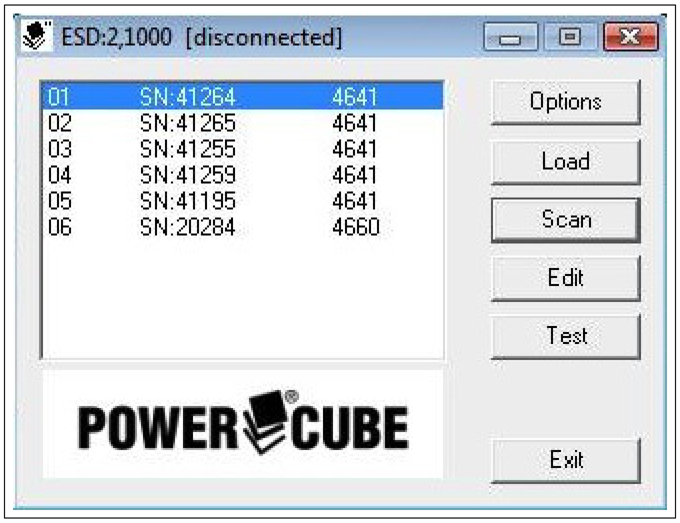
obotics\PowerCube Software\PowerCube.ini

Standard: ESD:2,1000

Other Options: RW.

Figure 11.16: "Options" screen for PowerCube

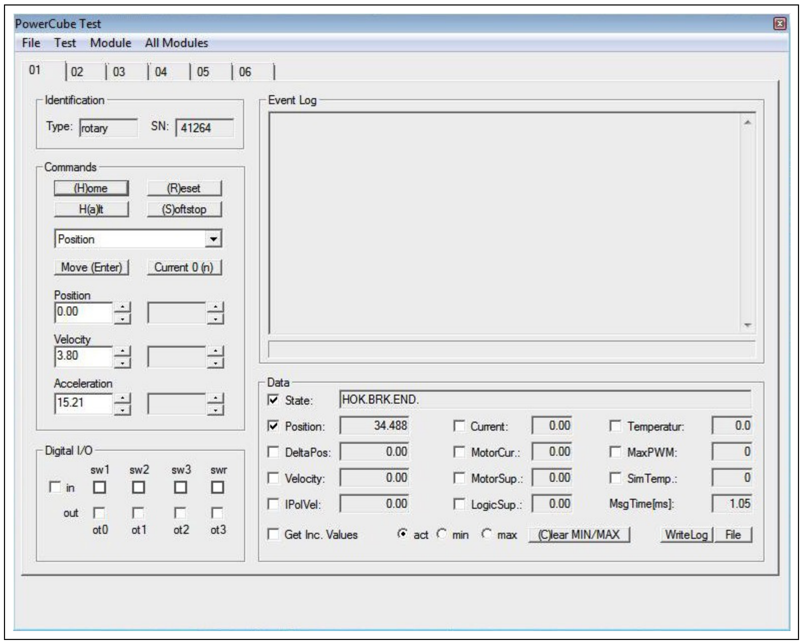
- Press "Scan" again. If nothing appears, go back to "Options" and repeat the process for changing the port number. If that does not work, change the communication options entirely, but remember to use the port numbers you recorded from Device Manager. If none of the previous works, please contact Schunk.



```
01   SN:41264   4641
02   SN:41265   4641
03   SN:41255   4641
04   SN:41259   4641
05   SN:41195   4641
06   SN:20284   4660
```

Figure 11.17: Listed modules after pressing "Scan"

- When the port number is correct and "Scan" is pressed, one should see a listing of six modules in the white area of the window. One can now press the "Test" button to test the movement of each individual module. There is no built-in self-collision avoidance so be careful of the joint angle values you command the joint to move to!



Type: rotary
SN: 41264
Position (mode): Position
Position: 0.00
Velocity: 3.80
Acceleration: 15.21

☑ State:	HOK.BRK.END.				
☑ Position:	34.488	☐ Current:	0.00	☐ Temperatur:	0.0
☐ DeltaPos:	0.00	☐ MotorCur.:	0.00	☐ MaxPWM:	0
☐ Velocity:	0.00	☐ MotorSup.:	0.00	☐ SimTemp.:	0
☐ IPolVel:	0.00	☐ LogicSup.:	0.00	MsgTime[ms]:	1.05

☐ Get Inc. Values [✓ act] min max

Figure 11.18: Test suite for PowerCube

For information regarding the position, velocity, and acceleration limits, as well as the other technical specifications such as PID gains for each module please refer to the documentation provided by Schunk.

Programming the Schunk 5DOF

To get started programming the robot with the PowerCube API, do the following:

- Open Visual C++ 2008 Express Edition
- In the top menubar, select "File" ➫ "New" ➫ "Project".
- Enter a name for your project. Change the location of the project to a place on the computer easily assessable to you. Press "OK".
- For the next screen, click "Next".
- On the Applications Settings screen, uncheck the option for "Precompiled header", check the option for "Empty project", then click "Finish".
- In the solution explorer, right-click the folder "Source Files" ➫ "Add" ➫ "New Item...". Select a C++ file, name it "main.cpp" then click "Add".
- With the screen open to your main.cpp file, modify the file to look like the following:

```
#include <iostream>

int main()
{
   return 0;
}
```

- We need to generate the debug folder so we can copy the PowerCube SDK's dynamically-linked library file (.dll) into that folder. To do that we must compile the program. In the top menubar, click: Build ➫ Rebuild Solution, or on your keyboard press [Ctrl]+[Alt]+[F7].

- Copy the PowerCube header file. Using Windows Explorer, find the folder labeled "amtec robotics". Inside, find and copy the file "m5apiw32.h".

- Use Windows Explorer to find you project folder on the computer. Inside you project folder, find the location of the main.cpp file, then paste the file "m5apiw32.h" in the same location.

- Back in VC++, in the Solution Explorer, right-click "Header Files" ➫ "Add" ➫ "Existing Item...". Navigate to the location of your main.cpp file. Once there, you will see the m5apiw32.h file there if you did the previous steps correctly. Select that file and click "Add".

- Locate the "amtec robotics" folder using Windows Explorer. Inside, find and copy the file "m5apiw32.dll".

- Locate your project folder using Windows Explorer. Inside your project there are two debug folders, find the one with your project's Incremental Linker File. The icon for this file resembles a paper with a "chain link" in the middle. Paste the .dll file from previous step into this folder.

- Back in VC++, right-click the project icon, select "Properties" ➫ "Linker" ➫ "General".

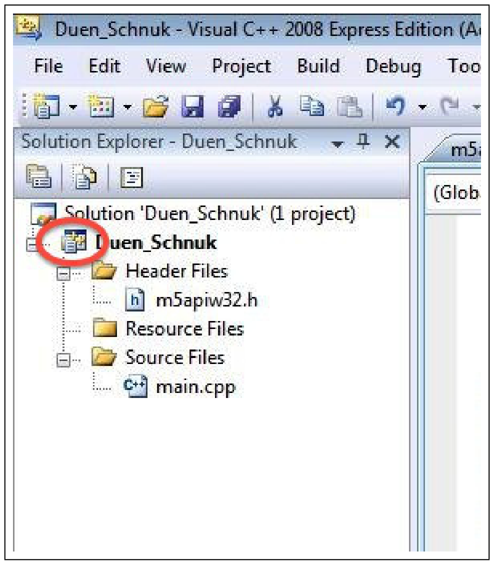
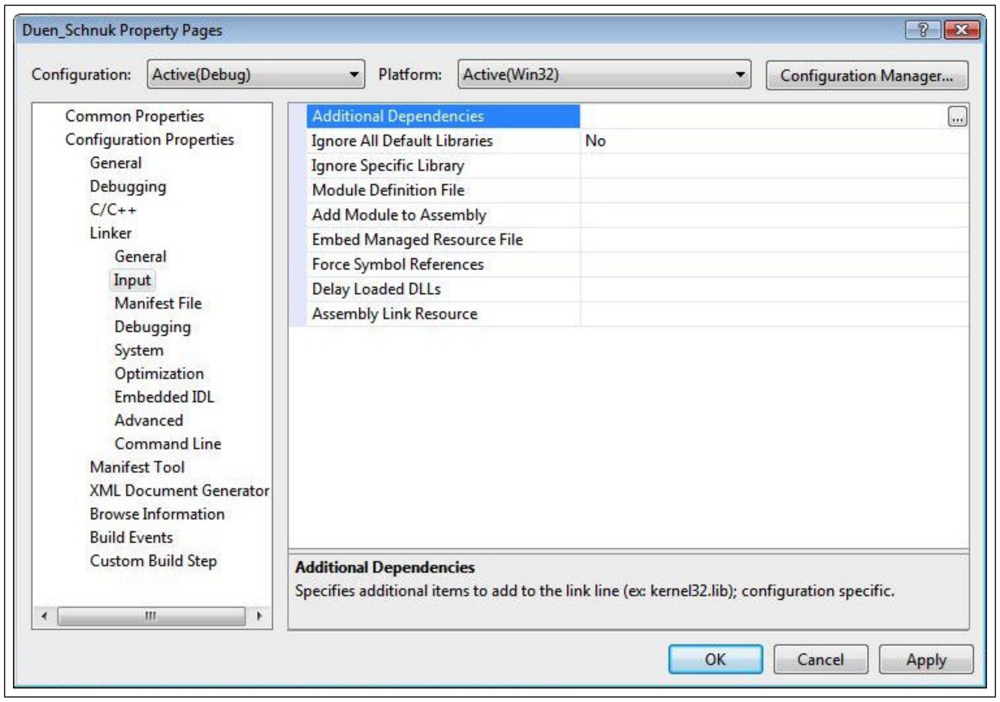

Duen_Schnuk - Visual C++ 2008 Express Edition (A
File Edit View Project Build Debug Too
Solution Explorer - Duen_Schnuk
Solution 'Duen_Schnuk' (1 project)
Duen_Schnuk
Header Files
m5apiw32.h
Resource Files
Source Files
main.cpp

Figure 11.19: VC++ project icon

- Select the field labeled "Additional Library Directories", and the press the "..." button that appears.

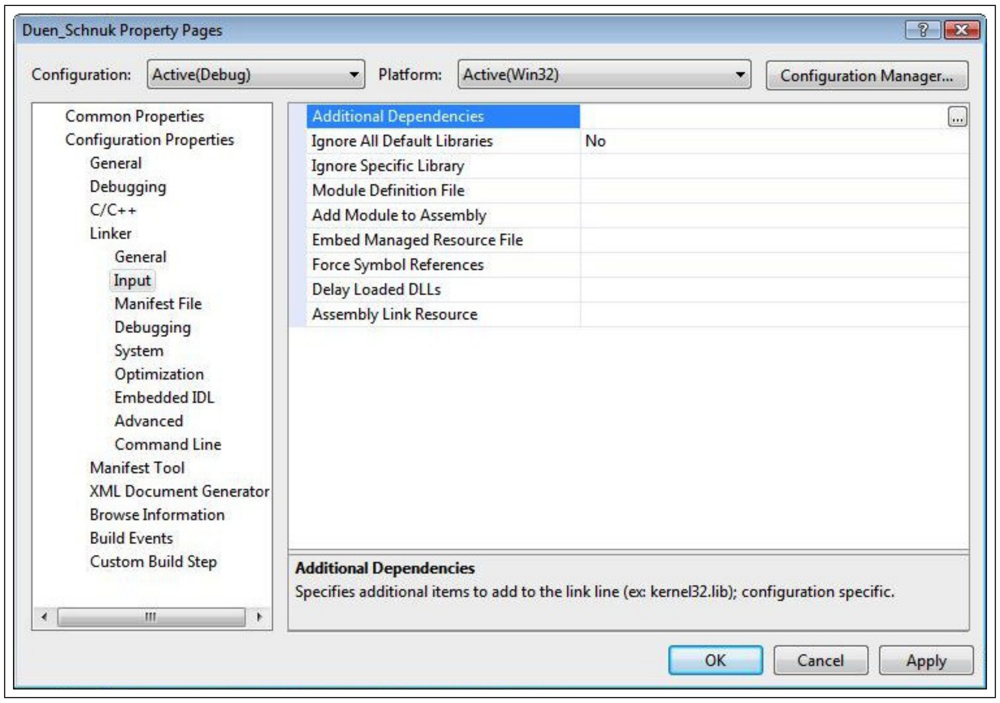


Additional Dependencies	
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

Figure 11.20: VC++ project icon

- Press the button with the folder icon, the press the "..." button that appears.
- Find the "amtec robotics" folder, find the "PowerCube Software" folder inside, select it, and then press the "Select" button. Add a "\" to the end of the path. Press "OK".
- In the Linker section, select "Input".
- Select the field "Additional Dependencies", then press the "..." button that appears.



Additional Dependencies	
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

Figure 11.21: Setting up the additional project dependencies

- In the white area, type "m5apiw32.lib". Press "OK".
- Now press "Apply", then press "OK".
- Modify the previous program in VC++ to look like the following:

```
#include <iostream>
#include "m5apiw32.h"
```

```
int main()
{
   return 0;
}
```

- Rebuild the solution as shown previously.

Now with everything in place, we are ready to begin programming the Schunk robot. Modify your program in VC++ to resemble the following program:

```
#include <iostream>
#include "m5apiw32.h"

int main()
{
   // device number
   int devID = 0;

   // communication type specifier
   const char comm[] = "ESD:2,1000";

   // open the Schunk device
   if(PCube_openDevice(&devID, comm) != 0)
   {
      (void)printf("Could not open device...\n\n")
      return EXIT_FAILURE;
   }
   else
      (void)printf("Schunk device is open.\n\n");

   // do stuff here...

   // close the device
   if(PCube_closeDevice(devID) != 0)
      (void)printf("Error occurred while closing device...\n");

   return 0;
}
```

The above program opens does several important things:

- Receives and stores the device ID in the variable **devID** and specifies which port to send the commands.
- The string "ESD:2,1000" is the same one from section on PowerCube software. The communication specifier from that section and the one you use in the program above needs to match. To do this, make the string in the above program match the one used in the PowerCube software.
- Attempts to open the robot device. If the device cannot be opened, the program quits. Otherwise, the device is opened and then closed with a call to **PCube_closeDevice(devID)**. *Every function in the PowerCube API returns zero on success, and a non-zero number otherwise.*

Rebuild the program. In the top menubar, click "Debug" ⇰ "Start without debugging", or on the keyboard press [Ctrl]+[F5] to run the program. Now, the above program will be modified to show how to initialize the rotary modules and the linear gripper module. A demonstration of how to synchronize the rotary modules, and command their motion will also be given. Modify the previous program to resemble the following program:

```
#include <iostream>
#include "m5apiw32.h"

#define PI 3.14

int main()
{
   // device number
   int devID = 0;

   // module ID numbers, need to be const
   const int module1 = 1;
   const int module2 = 2;
   const int module6 = 6;

   // module states
   unsigned long tempState = 0x0L; temporary variable for reading states
   unsigned long moduleState1 = 0x0L; // base joint
   unsigned long moduleState2 = 0x0L; // joint #2
   unsigned long moduleState6 = 0x0L; // gripper

   // communication type specifier
   const char comm[] = "ESD:2,1000";

   // open the Schunk device
   if(PCube_openDevice(&devID, comm) != 0)
   {
      (void)printf("Could not open device...\n\n")
      return EXIT_FAILURE;
   }
   else
      (void)printf("Schunk device is open.\n\n");

   // set up rotary modules to move at the same time, e.g., in sync
   // set module 1 to be M3 compatible
   if(PCube_setConfig(devID, module1, CONFIGID_MOD_M3_COMPATIBLE) != 0)
   {
      (void)printf("Could not properly configure robot.\n\n");
      return EXIT_FAILURE;
   }
   // set the motion for module 1 to be in sync with other rotary modules
   if(PCube_setConfig(devID, module1, CONFIGID_MOD_SYNC_MOTION) != 0)
   {
      (void)printf("Could not properly configure robot.\n\n");
      return EXIT_FAILURE;
   }

   // set module 2 to be M3 compatible
   if(PCube_setConfig(devID, module2, CONFIGID_MOD_M3_COMPATIBLE) != 0)
   {
      (void)printf("Could not properly configure robot.\n\n");
      return EXIT_FAILURE;
   }
   // set the motion for module 2 to be in sync with other rotary modules
```

```
if(PCube_setConfig(devID, module2, CONFIGID_MOD_SYNC_MOTION) != 0)
{
   (void)printf("Could not properly configure robot.\n\n");
   return EXIT_FAILURE;
}

// send position commands to joints 1 and 2
// clear messages from modules
if(PCube_resetAll(devID) != 0)
{
   (void)printf("Could not reset modules prior to operation.\n\n");
   return EXIT_FAILURE;
}
// base joint
if(PCube_movePos(devID, module1, PI/6) != 0)
{
   (void)printf("Could not move joint 1 to position.\n\n");
   return EXIT_FAILURE;
}
// joint 2
if(PCube_movePos(devID, module2, PI/4) != 0)
{
   (void)printf("Could not move joint 2 to position.\n\n");
   return EXIT_FAILURE;
}
// start motion for all modules
if(PCube_startMotionAll(devID) != 0)
{
   (void)printf("Could not start motion for all modules.\n\n");
   return EXIT_FAILURE;
}
// get the current state for the modules in motion
// get module 1 state
if(PCube_getModuleState(devID, module1, &tempState) != 0)
{
   (void)printf("Could not obtain module 1 state.\n\n");
   return EXIT_FAILURE;
}
// extract state
moduleState1 = tempState & STATEID_MOD_MOTION;
// get module 2 state
if(PCube_getModuleState(devID, module1, &tempState) != 0)
```

```
{
   (void)printf("Could not obtain module 2 state.\n\n");
   return EXIT_FAILURE;
}
// extract state
moduleState2 = tempState & STATEID_MOD_MOTION;
// wait until the robot is done moving for initialization
while(moduleState1 == STATEID_MOD_MOTION ||
 moduleState2 == STATEID_MOD_MOTION)
{
   // get the current state for the modules in motion
   if(PCube_getModuleState(devID, module1, &tempState) != 0)
   {
      (void)printf("Could not obtain module 1 state.\n\n");
      return EXIT_FAILURE;
   }
   // extract state
   moduleState1 = tempState & STATEID_MOD_MOTION;
   // get the current state for module 2
   if(PCube_getModuleState(devID, module2, &tempState) != 0)
   {
      (void)printf("Could not obtain module 2 state.\n\n");
      return EXIT_FAILURE;
   }
   // extract state
   moduleState2 = tempState & STATEID_MOD_MOTION;
}

// configure the gripper
if(PCube_homeModule(devID, module6) != 0)
{
   (void)printf("Could not home gripper module.\n\n");
   return EXIT_FAILURE;
}

// get the state of the gripper module
if(PCube_getModuleState(devID, module6, &tempState) != 0)
{
   (void)printf("Could not get the current state for gripper.\n\n");
   return EXIT_FAILURE;
}
// extract state
```

```
    moduleState6 = tempState & STATEID_MOD_MOTION;

    // while the gripper fingers are in motion
    while(moduleState6 == STATEID_MOD_MOTION)
    {
       // get the state of the gripper module
       if(PCube_getModuleState(devID, module6, &tempState) != 0)
       {
          (void)printf("Could not get the current state for gripper.\n\n");
          return EXIT_FAILURE;
       }
       // extract state
       moduleState6 = tempState & STATEID_MOD_MOTION;
    }

    // reset the gripper after homing
    if(PCube_resetModule(devID, module6) != 0)
    {
       (void)printf("Could not reset gripper module.\n\n");
       return EXIT_FAILURE;
    }

    /**** do stuff here ****/

    // close the device
    if(PCube_resetAll(devID) != 0)
       (void)printf("Error occurred while clearing messages for closing.\n\n");

    if(PCube_closeDevice(devID) != 0)
       (void)printf("Error occurred while closing device...\n");

    return 0;
}
```

This program adds a significant amount of code to its predecessor, in summary:

- Declared and initialized variables for module ID numbers for each module to be used. These variables should be of type **const int**.
- Declared and initialized variables for the states of each module. These should be of

type **unsigned long**, as the PowerCube constants for the module states are of type **unsigned long**.

- Two calls are made to **PCube_setConfig(int devID, int modID, unsigned long configID)** for each rotary module, one call for each of the necessary configuration constants. These calls are necessary for configuring the modules to move synchronously.

- Commands to move to a given position are made by calling **PCube_movePos(int devID, int modID, float pos)** for each module, then tell the modules to *start moving now* by calling **PCube_startMotionAll(devID)**, this is an important point the reader needs to understand.

- For each module, the program queries the robot for the states (each module can have more than one state at a time), which is represented by a single number, the number is read into a temporary variable and then AND operator is used to extract the state of interest.

- The program then waits in a loop for the rotary modules to finish moving.

- Next, the program calls **PCube_homeModule(int devID, int modID)** to "home" or calibrate the linear gripper. This is necessary because the gripper does not have an absolute encoder.

- Just as with previous modules, the state of the gripper is obtained and the program waits for the gripper to finish moving.

Rebuild the program. Run the program. All other necessary programming would occur between the "do stuff here" and "close the device" comments. To get more details on these and other PowerCube API functions, please refer to the Schunk documentation "Programmers guide for PowerCube" and the "m5apiw32.h" file for the function prototypes. Additional notes for consideration:

- The linear gripper has a maximum opening of $60.0mm$ and a *real minimum* of $1.0mm$. If you put in a value less than $1.0mm$, the gripper will get an error and may shut down. The only way to clear the error may involve removing power and then reapplying power to the whole robot.

- Positions for the rotary modules must be in radians, positions for the linear gripper must be in meters.

- When commanding a position for the linear gripper, use **PCube_moveRamp(int devID, int modID, float pos, float vel, float accel)**.

APPENDIX 2: IRB Approval Documentation

FSU Behavioral Consent Form
Usability in Teleoperation and Control.

You are invited to be in a research study of user operation for teleoperation and control. You were selected as a possible participant at random. We ask that you read this form and ask any questions you may have before agreeing to be in the study.

This study is being conducted by Brandon Allen Charles Peters, under the direction of Dr. Carl A. Moore, Jr., and Dr. Rodney G. Roberts at the FAMU-FSU College of Engineering.

Background Information:

This study serves to determine the best mode of operation for a teleoperation system given user feedback.

Procedures:

If you agree to be in this study, we would ask you to do the following things:

First, there will be no audio or video recording of test subjects in this study. Data collected in the experiments will come from queries of the master controller position and velocity and the slave manipulator position and velocity in the control program, time, and ease-of-use. You will be asked to control and move a robotic arm around on a table and cut color-coded wires using two different control schemes. You will have to do this task in a specified order given by the facilitator. Once finished, the facilitator will give you a form on which you will evaluate the ease-of-use of each control scheme. The facilitator will not view these forms until all experiments are complete.

Risks and benefits of being in the Study:

The study has some risks. There is a small chance of being hit by the robotic arm. The participant may also be exposed to being hit by the wire-holders if the robotic arm knocks one or more of them into the air. These things are unlikely to happen but the risks remain. There are no benefits to the participants of this study, however there are benefits to society mainly through the robotics community. The main benefit is the possibility of an easier method of moving a robotic arm in its environment with less work and input on the user's part, greatly reducing user mental workload and physical fatigue. This will lead to easier-to-use wartime systems, surgical systems, and robotic toxic cleanup systems.

Compensation:

Participants in this study will not be compensated by money or any other material means.

Confidentiality:

The records of this study will be kept private and confidential to the extent permitted by law. In any sort of report we might publish, we will not include any information that will make it possible to identify a subject. Research records will be stored securely and only researchers will have access to the records.

Voluntary Nature of the Study:

Participation in this study is voluntary. Your decision whether or not to participate will not affect your current or future relations with the University. If you decide to participate, you are free to not answer any question or withdraw at any time without affecting those relationships.

Contacts and Questions:

The researcher conducting this study is Brandon A. C. Peters. You may ask any question you have now. If you have a question later, you are encouraged to contact him at the National High Magnetic Field Laboratory, 1800 East Paul Dirac Dr, Tallahassee, FL, 850-412-5743, His advisor is Dr. Carl A. Moore, Jr., and can be contacted at the National High Magnetic Field Laboratory, 1800 East Paul Dirac Dr, Tallahassee, FL, 850-412-5743,

If you have any questions or concerns regarding this study and would like to talk to someone other than the researcher(s), you are encouraged to contact the FSU IRB at 2010 Levy Street, Research Building B, Suite 276, Tallahassee, FL 32306-2742, or 850-644-8633, or by email at humansubjects@magnet.fsu.edu.

You will be given a copy of this information to keep for your records.

Statement of Consent:

I have read the above information. I have asked questions and have received answers. I consent to participate in the study.

Signature	Date
Signature of Investigator	Date

From: Human Subjects <humansubjects@magnet.fsu.edu>
Subject: Use of Human Subjects in Research - Approval Memorandum
Date: January 4, 2012 4:21:54 PM EST
To:
Cc:

1 Attachment, 20 KB

Office of the Vice President For Research
Human Subjects Committee
Tallahassee, Florida 32306-2742
(850) 644-8673 · FAX (850) 644-4392

APPROVAL MEMORANDUM

Date: 1/4/2012

To: Brandon Peters

Dept.: COLLEGE OF ENGINEERING

From: Thomas L. Jacobson, Chair

Re: Use of Human Subjects in Research
Usability of Different Command Strategies for Teleoperation

The application that you submitted to this office in regard to the use of human subjects in the proposal referenced above have been reviewed by the Secretary, the Chair, and one member of the Human Subjects Committee. Your project is determined to be Expedited per per 45 CFR § 46.110(7) and has been approved by an expedited review process.

The Human Subjects Committee has not evaluated your proposal for scientific merit, except to weigh the risk to the human participants and the aspects of the proposal related to potential risk and benefit. This approval does not replace any departmental or other approvals, which may be required.

If you submitted a proposed consent form with your application, the approved stamped consent form is attached to this approval notice. Only the stamped version of the consent form may be used in recruiting research subjects.

If the project has not been completed by 1/1/2013 you must request a renewal of approval for continuation of the project. As a courtesy, a renewal notice will be sent to you prior to your expiration date; however, it is your responsibility as the Principal Investigator to timely request renewal of your approval from the Committee.

You are advised that any change in protocol for this project must be reviewed and approved by the Committee prior to implementation of the proposed change in the protocol. A protocol change/amendment form is required to be submitted for approval by the Committee. In addition, federal regulations require that the Principal Investigator promptly report, in writing any unanticipated problems or adverse events involving risks to research subjects or others.

By copy of this memorandum, the Chair of your department and/or your major professor is reminded that he/she is responsible for being informed concerning research projects involving human subjects in the department, and should review protocols as often as needed to insure that the project is being conducted in compliance with our institution and with DHHS regulations.

This institution has an Assurance on file with the Office for Human Research Protection. The Assurance Number is FWA00000168/IRB number IRB00000446.

Cc: Carl Moore, Advisor
HSC No. 2011.7541

7541.pdf (20 kB)

REFERENCES

[1] Sung Ho Ahn, Byung Suk Park, and Ji Sup Yoon. A teleoperation position control for 2-dof manipulators with control input saturation. *ISIE*, pages 1520 – 1525, 2001.

[2] Mark T. Alise. *Expansion and Implementation of the Wave Variable Method in Multiple Degree-of-Freedom Systems*. PhD thesis, FAMU-FSU College of Engineering, 2007.

[3] James W. Dally, William F. Riley, and Kenneth G. McConnell. *Instrumentation for Engineering Measurements*. John Wiley and Sons, 1984.

[4] Ildar Farkhatdinov and Jee-Hwan Ryu. Hybrid position-position and position-speed command strategy for the bilateral teleoperation of a mobile robot. *International Conference on Control, Automation, and Systems 2007*, pages 2442 – 2447, October 2007.

[5] Henning Hayn and Dieter Schwarzmann. Control concept for a hydraulic mobile machine using a haptic operating device. *2009 Second International Conferences on Advances in Computer-Human Interaction*, pages 348 – 353, 2009.

[6] Alejandro Jarillo-Silva, Omar A. Dominguez-Ramirez, Vicente Parra-Vega, and J. Patricio Ordaz-Oliver. Phantom omni haptic device: Kinematic and manipulability. *2009 Electronics, Robotics and Automotive Mechanics Conference*, pages 193 – 198, 2009.

[7] Carl A. Moore Jr. *Design, Contruction, and Control of a 3-Revolute Arm Cobot*. PhD thesis, Northwestern University, 2001.

[8] Brandon A. C. Peters, Carl A. Moore Jr., and Rodney G. Roberts. Teleoperaton setup for wire-cutting application in the field. *2012 IEEE Southeast Symposium on System Theory*, March 2012.

[9] Agostino De Santis, Alin Albu-Schaffer, Christian Ott, Bruno Siciliano, and Gerd Hirzinger. The skeleton algorithm for self-collision avoidance of a humanoid manipulator. *Advanced Intelligent Mechatronics, 2007 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, September 2007.

[10] Fumi Seto, Kazuhiro Kosuge, and Yasuhisa Hirata. Self-collision avoidance motion control for human robot cooperation system using robe. *Intelligent Robots and Systems 2005, IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3143 – 3148, 2005.

[11] Gang Song and Shuxiang Guo. Development of a novel tele-rehabilitation system. *Proceedings of the 2006 IEEE International Conference on Robotics and Biomimetics*, pages 785 – 789, December 2006.

[12] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley and Sons, Hoboken, New Jersey, 2006.

[13] Eduardo Veras, Karan Khokar, Redwan Alqasemi, and Rajiv Dubey. Scaled telerobotic control of a manipulator in real time with laser assistance for adl tasks. *Mechatronics and its Applications, 2009. International Symposium on Mechatronics and its Applications*, March 2009.

BIOGRAPHIC SKETCH

Brandon Allen Charles Peters was born in Tallahassee, FL, on the 16th day of February, in the year 1985. After graduating from high school, he attended Florida State University and earned his Bachelor's of Science in Electrical Engineering in December 2008. He returned to the college in August of 2009 to complete a graduate degree in electrical engineering, completing his Master's of Science in Electrical Engineering in the Spring of 2012.