**Robotic Wheelchair Central Computer – Collaboration of Hardware Security Research and Robosense VIP Teams**

**Wesley Schon**

**Georgia Institute of Technology**

**Vertically Integrated Projects**

**Secure Hardware Team**

**04 December, 2014**

**Introduction**

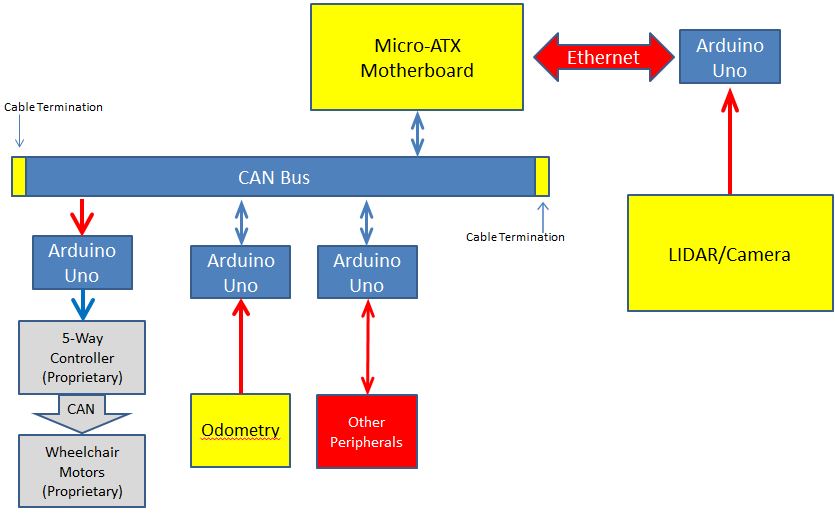
This paper details the hardware and software specifications for the central computer of an ongoing project to produce an autonomous wheelchair. Information will be included on the project background, goals, construction of the computer, the interfacing of the computer with peripherals, compilation/installation of specific drivers, the operating system, and nuances of communicating with the wheelchair motors.

**Background**

At the start of this phase of the project, a number of hardware items had already been obtained, including a Micro-ATX Intel motherboard, Intel Core i3 processor, stock fan, and a separate CPU fan, two 8GH DDR3 DIMMs (Dual Inline Memory Modules), A Samsung 250 GB SDD, a 500W ATX 5V desktop power supply, and a 250W output, 6V to 30V wide input DC-DC car PC power supply. For the CAN communication, CAN hardware from PEAK Systems had been purchased to interface with both the USB and PCI ports on the motherboard. In addition, 9-wire cable, resistors, 9-pin shields, clamps, two Arduino Uno microprocessors, and two Seeed CAN shields for Arduino were provided. The software platform, Linux Ubuntu 14.0x and ROS Indigo had been decided upon, as well as the overall architecture, shown in Figure 1. Over the course of the semester, I worked closely with Dan Pettinger, another VIP Hardware Security Team member this semester. His report more closely covers certain aspects of the project, such as CAN protocols and Arudino programming.

**Goals**

This project had several goals. The overall project goal for the Robosense Wheelchair team is to produce an autonomous robotic wheelchair which makes decisions based on people that it detects using computer vision, their position, and their actions. The goal of the central computer team was to build a computer to serve as the main control system for this robot, to obtain and build the necessary software platform for this computer, and to facilitate its ability to communicate with other parts of the robot. The overall current target architecture for this robot is shown in Figure 1. Side goals for this project included the design, construction, and testing of a physical CAN bus, and the analysis of the wheelchair’s motor control system.



**Figure 1.** Target architecture for robotic wheelchair. Completed parts are shown in blue, incomplete parts in yellow, and future goals in red.

**Computer Hardware Installation**

# When constructing a computer, it is important to touch grounded items often to dissipate any static charge accumulated on the body often to avoid damaging fragile components. We started with the micro-ATX motherboard, keeping it elevated throughout the process on the anti-static wrap in which it was originally packaged. We first installed the CPU. The CPU cover is located near the middle of the board. It can be lifted by unhinging the metal lever on the side and lifting that. The protective plastic that comes with the motherboard can be snapped out easily, and can be stored away (or discarded). While handling the processor, touch only the sides. The pins and notches on the processor must be lined up with those in the shallow processor bay on the motherboard, and the processor should be placed into the bay gently. The CPU cover is swung back over the processor and locked into place by pushing the metal lever back down and into its original place (this may take a little bit of force). Next, the CPU fan with thermal paste can be placed directly over the CPU and locked in place by pushing the pins on each corner of the fan down through their respective holes on the motherboard. The wires from this fan can go directly into the motherboard’s fan header. On this motherboard we used the SYS-FAN header nearest the CPU. The motherboard manual provides details for all the necessary SATA connections. Next, we installed the RAM. There are four RAM ports on the motherboard and we used ports 1 and 3. Each stick of RAM needs only to be lined up along the port and inserted. Clamps on either end of the port are then closed to hold the RAM in place. For the desktop power supply and the hard drive, we plugged the cables from each into the necessary motherboard headers as directed in the motherboard manual. All extra cables from the power supply were bundled using twisty-ties to keep the workspace organized.

# After the computer had been constructed, it was turned on by briefly placing a screwdriver between the two power pins on the F\_PANEL header on the motherboard. This shorts the pins the same way pressing a power button would. The computer powered on successfully, and Linux Ubuntu was installed from a bootable USB stick. The setup at the time had two drawbacks. First, keeping a screwdriver on-hand to short the F\_PANEL pins to power the computer was impractical. Second, the exposed parts and wires presented a security risk, because it would be easy for anyone in the area to inadvertently damage the components. A picture of the computer before mounting in the case is shown in Figure 2. For these reasons, we purchased a mini-tower case to contain all the components. The motherboard screwed into a platform in the case at several points that also served to ground the board. Later on, when the CAN bus was being set up and tested, a PCI-CAN card was installed into the PCI slot on the motherboard facing the back panel of the tower case.

# 

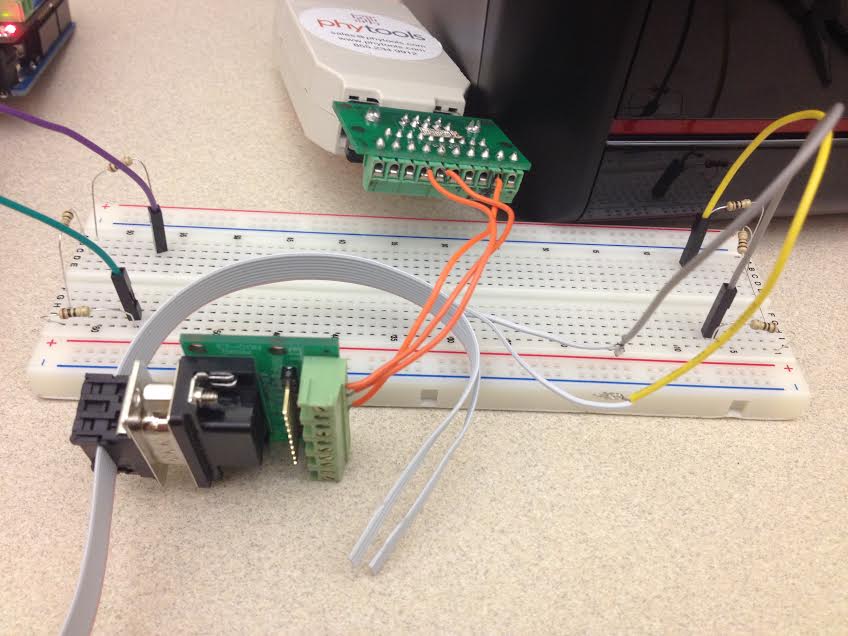
# Figure 2. Central Computer before installation into tower case.

# Interfacing Central Computer with Peripherals

In order for the central computer to communicate quickly and robustly with various peripherals (e.g. motor control and odometry) a CAN (Controller Area Network) bus has been implemented. Two different interfaces have been attached and tested with the central computer, namely USB-CAN and a PCI-CAN. Compilation of the drivers for this hardware is covered in the section **Compilation of PCAN Drivers**. On the peripheral side, Arduino UNO microcontrollers using CAN shields from Seeed Studio serve as interpreters which read CAN messages and communicate directly with peripherals. The wheelchair also has its own proprietary CAN bus, which connects directly to a 5-way controller. This controller can be communicated with via an Arduino UNO, as is explained in the section **Driving the Motors of the Wheelchair**.

**Controller Area Network (CAN) Bus**

The physical CAN bus consists of a 9-wire cable terminated at each end by a resistance equal to the characteristic impedance of the cable. At each node, a female 9-pin shield with conductive metal teeth bites down on the cable, allowing any interface with a 9-pin serial male connector to access the bus. Standardization of CAN-High and CAN-Low signals is accomplished in the current setup using two 9-pin shields between the bus line and central computer, and connecting wires from pin 3 on the bus-side to pin 7 on the computer-side, and pin 5 on the bus-side to pin 2 on the computer-side. This is shown in Figure 3.



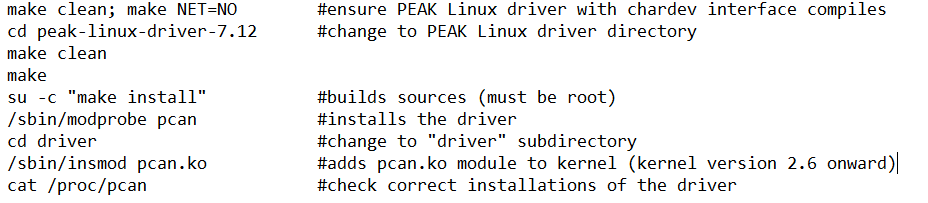
**Figure 3.** CAN shields shifting the bus pins to the USB-CAN pins. An additional wire carries GND.

**Operating System**

The installation of the Linux Ubuntu operating system and ROS are both very straightforward. In this case, we used Linux Ubuntu 14.04.1 LTS and ROS Indigo. In order to install Ubuntu, the computer must be booted from a bootable USB drive containing Ubuntu from the BIOS (the only accessible part of the system without an operating system). After that, it is sufficient to follow the on-screen prompts. Once Ubuntu has been installed, ROS can be installed. Ros.org provides a step-by-step guide for doing this, which is what we used in this project. An internet connection is required to install ROS. Root permissions are useful to perform these installation steps but not necessary unless otherwise indicated in the guide (for example, while setting up repositories). This can be gained by running “sudo su” or “sudo su root” from the command line and entering the system password. If root access is not desired, the “sudo” prefix to a command allows for commands to be run as root, and requires the system password to be entered.

**Compilation of PCAN Drivers**

PCAN (PEAK-CAN) driver installation is necessary to get the CAN hardware discussed in **Interfacing Central Computer with Peripherals** working. A .tar file containing all of the necessary files can be found on the PEAK-Systems website. After this folder has been downloaded from the website or imported from another system via USB drive, email, CD, or any other method, the drivers can be installed with or without an internet connection. It is important to note that there exist two separate versions of the driver, called chardev and netdev. The netdev version interfaces with the kernel via a so-called Socket-CAN framework, and will not work properly with our setup for unknown reasons, causing issues with the system’s ability to receive messages. The first line of the instructions in Figure 4 ensures that the chardev interface is used, rather than the netdev. Depending on the system, an error message may be thrown during the installation stating that “popt.h could not be found.” If this occurs, it is necessary to get popt.h for your system before proceeding. This is accomplished by running “apt-get install popt” from the command line as the root user. The following Figure details the steps necessary to install the drivers.

****

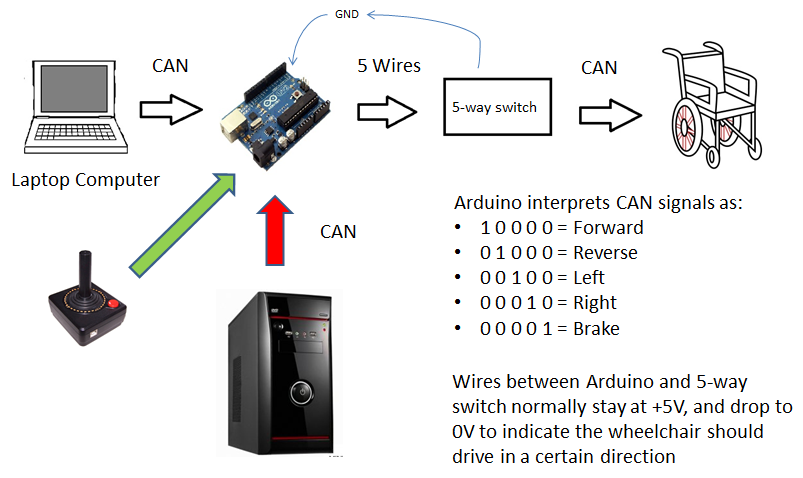
**Figure 3.** Installation of the PCAN drivers with chardev interface

After the driver package has been installed, it is necessary to test it. The package contains drivers for both PCI and USB CAN adapters. In order to test, a Windows laptop and Arduino UNO with a CAN shield works very well. Using the CAN software already available for use with Arduino UNO, messages can be sent and received by the Windows laptop without difficulty. In the Linux system, there are two functions called transmitest and receivetest that can be used to test proper installation, located in a subdirectory of the installation files taken from the PEAK-Systems website. By navigating to that subdirectory, the tests can be run. If the Arduino is sending a message, reception on the Linux system can be tested by running “receivetest f=/dev/pcan32.” If the Arduino is receiving messages, transmission can be tested by running “transmitest hello.txt f=/dev/pcan32.” A .txt file called “hello.txt” is located in the same directory and contains a standard CAN message that can be altered. The pcan32 can also be changed to another device if necessary; here, it represents the USB-PCAN hardware. A list of all available CAN devices can be seen by running “cat /proc/pcan” from the command line, and “ls –l /dev/pcan\*” lists all the device nodes.

**Driving the Motors of the Wheelchair**

The wheelchair’s motors are connected directly to a 5-way controller via a proprietary CAN bus. This 5-way controller can be controlled by placing a shield over the serial port and routing wires to it from an Arduino UNO. Pins 1-5 coming from the controller can be connected to pins 2-6 on the UNO. Other pin configurations may work if needed, but changes to the current Arduino code would need to be made. The 5-way controller has five input pins that stay at +5V. In order to “activate” these pins, thus causing the 5-way controller to drive the motors, the pins must be grounded. This has been accomplished by having the Arduino drive its pins connected to the 5-way controller pins to LOW. The Arduino has also been set to reference the controller’s GND.

Initially, the Arduino used inputs from a joystick and relayed them to its own outputs to the controller in order to drive the wheelchair. We set this up first to allow us to troubleshoot problems on the controller side without having to worry about CAN errors. After this was accomplished, the Arduino was connected to a Windows laptop over CAN and would interpret the incoming CAN signals to drive the controller, and thus, the wheelchair. A diagram of the setup, including the CAN signals sent from the laptop to the Arduino, is shown in Figure 5. Some irregularities have been found with the way the 5-way controller drives the motors, and in general, the 5-way controller has worked inconsistently. The signals coming from the Arduino need to account for these nuances and should be fine-tuned for the system. The signals going to the 5-way controller on pins 1-5 correspond to Forward, Reverse, Left, Right, and Brake respectively. These pins receive signals from pins 2-6 on the Arduino, in that same order.



**Figure 5.** Driving the wheelchair over CAN. The joystick was a preliminary setup, the PC a future goal.

**Future Work**

Currently, the wheelchair has only been driven over CAN using a windows laptop. The system consisting of the Arduino UNO and 5-way controller is certainly adaptable to be used with Linux Ubuntu. The current implementation of CAN on Ubuntu only allows for reception or transmission using the receivetest and transmitest programs included in the installation of the CAN drivers. This means that the commands being sent from the computer cannot be quickly changed; it is necessary to edit a text file and rerun the program in order to change the command. In the future, the computer should have custom send and receive functions built to be used with the CAN hardware. These functions can be implemented in a larger program, which will allow for quicker control on the user’s part. Once this is done, the wheelchair can be driven by the central computer in the same manner as it has been by the windows laptop. In addition, the Arduino code controlling the wheelchair’s movement via the 5-way switch can be refined to provide the user with greater control over the wheelchair’s response. As more groups associated with the wheelchair project become ready to incorporate their projects into the greater implementation of the system, it will be necessary to work closely with them to link their projects with the central computer. This will require the creation of an addressing system for the CAN bus, as well as the addition of more nodes to the existing bus. Once more nodes have been added and more data is being sent over the bus, its ability to correctly arbitrate should be closely monitored and tested, as to this point arbitration has not been necessary. In addition, size and efficiency considerations should be made. Ideally, the number of microcontrollers and wires used should be reduced as much as possible. Multitasking on the microcontrollers might be necessary if their processing power allows.

**References**

[1] (2002-2007) Peak-System- Technik GmbH and Klaus Hitschler. PCAN-driver for Linux (Online) Available: <http://www.peak-system.com/fileadmin/media/linux/files/Installation-en_6.x.pdf>

[2] (2014) Open Source Robotics Foundation. ROS.org (Online) Available: <http://wiki.ros.org/indigo/Installation/Ubuntu>

[3] (2014) Dan Pettinger (VIP Secure Hardware student from same semester). Report and Presentation. Vertically Integrated Projects, Georgia Institute of Technology.

[4] (2014) Wesley Schon. Robosense Central Computer (PowerPoint). Included with this report.