6-2012

The Advanced Educational Robot

Calder Phillips-Grafflin
Union College - Schenectady, NY

The Advanced Educational Robot

By

Calder Phillips-Grafflin

****

Submitted in partial fulfillment
of the requirements for
Honors in the Department of Electrical and Computer Engineering

UNION COLLEGE

May, 2012

ABSTRACT

PHILLIPS-GRAFFLIN, CALDER The Advanced Educational Robot

ADVISOR: JOHN SPINELLI

Existing literature in the field of computer science education clearly demonstrates that robots can be ideal teaching tools for basic computer science concepts. Likewise, robots are an ideal platform for more complicated CS techniques such as evolutionary algorithms and neural networks. With these two distinct roles in mind, that of the teaching tool and that of the research tool, in collaboration with customers in the CS department we have developed a new robotics platform suitable for both roles that provides higher performance and improved ease-of-use in comparison to the robots currently in use at Union.

We have successfully designed and built a medium-sized robotics platform for classroom and research use that provides better maneuverability, increased flexibility, and is easier to use than commercial equivalents at significantly lower cost. In particular, our robot provides a platform with human-level mobility suitable for use in human-machine interaction (HMI) research and testing. Using a combination of easily available off-the-shelf parts, newly available sensors, and open-source software, we have built a platform that is both easy enough for beginners to use but also powerful enough for advanced users to customize and adapt to their specific needs.

Contents

Figures

Introduction

Existing robots used in educational environments fall into two categories: simple robotics kits and commercial robotics development platforms. The first category ranges from simple devices like Parallax's Scribbler to the more comprehensive Lego NXT ecosystem. These robots are cheap, relatively easy to use, and (in the case of Lego or Vex systems) relatively flexible in the ability to add additional sensors and actuators. However, these systems lack the kind of processing power necessary to process video input in real time or to run more complicated control algorithms. In most cases, such functionality must be implemented through the use of a host computer, which imposes sharp limits on portability and autonomy. [6][8]

In contrast, commercial robots range from comparatively simple devices like Mobile Robots' Pioneer series all the way up to complete development platforms such as Willow Garage's PR2. These robots almost universally have on-board computers capable of video processing and navigation, and provide nearly complete sensor suites for use in development. However, these robots often trade flexibility for reliability, with limited means for expansion or component replacement. In particular, the popular Pioneer series robots are designed to last years in educational settings, but are relatively limited in their ability for expansion (particularly true when using the on-board computer). In addition, these commercial robots are not always affordable; while simpler systems like a Pioneer start at approximately $4000 US, more flexible and complex platforms range up to $400,000 US in the case of the PR2. These are by no means affordable for educational use, especially if multiple units are desired. Thus, the ideal platform for educational robotics combines strong flexibility with high computational power and relatively low cost. [6][8]

Our project is the design and construction of a medium sized (approximately 40 kg when fully loaded) mobile robot for educational and research use. This robot is designed to provide a strong baseline chassis with sensors, actuators and necessary software to allow for easy integration with existing robotic equipment, software and curricula. In particular, it is designed

for use in intermediate and advanced undergraduate robotics courses and undergraduate research projects in robotics. The following descriptions from professors in the Computer Science department illustrate the characteristics needed for this:

> An ideal teaching robot for me would be mobile, robust, and capable of running the Robot Operating System (ROS). Mobility is important because most interesting problems in AI and robotics appear when robots leave "staged" labs and enter the real world. Robustness is important not only for any number of obvious reasons (students will be driving this robot into walls a lot, and I would like it to last longer than a week), but also because I'd like to use this as a platform for embodied evolution (EE) which requires hours and hours of unsupervised robot learning. ROS is an extremely versatile programming environment for teaching robotics - it is simple enough that students can write a module in a matter of minutes, and complex enough to be used in very expensive industrial robot systems. – John Rieffel, CS

> I am interested in seeing a robotic platform designed at Union, for use in both research and teaching. One of my interests is in social robotics. For a robot to be social, it has several basic requirements. First it needs to be able to navigate in human-scale environments. This means that it requires sufficient sensing capability to be able to perform both localization and object avoidance. In turn, this requires a platform capable of controlled maneuvering, and carrying a payload (processing power) that can support the sensing required.
>
> Second, it needs to be a minimum height to promote social interaction, via voice or gesture. Again, to support these requirements, the platform should be able to support an appropriate payload, including microphones, speakers and the necessary processing power. The platform as proposed [our project] is extensible, such that adding components (an extending head, for example) should be possible.
>
> Toward the end of this academic year, I am proposing to develop, in conjunction with upper level students, a toolkit for creating voice interfaces. Ultimately, as a test of such a system, it would be

> useful to deploy an interface created using this toolkit on a roving platform [our project] capable of navigating the computer science department. This would be a great continuing platform to experiment with both fundamental robotics principles (control, navigation, etc.) and interaction in a unified manner. – Nick Webb, CS

In addition to hardware design, this project involves the creation of the necessary software for complete integration between the robot and existing production-quality robotics software such as Willow Garage's ROS (Robotic Operating System) and Microsoft's MRDS. The proposed robot will be less expensive than a Pioneer while offering superior mobility and flexibility. In addition, we plan on making the robot accessible to everyone by using only open-source components whenever possible and by publishing the final plans on the Internet for anyone to use and adapt. Doing so allows other users, both at Union and at other schools, the ability to freely use and modify the design to suit their needs.

Background

Design Features

Previous researchers have attempted to define the ideal features of an educational robot, with Weiss and Overcast identifying the following criteria as essential: platform flexibility and extensibility, quality and durability of hardware, quality of documentation and teaching resources, availability of sample code, continuing developer activity, total system cost, SDKs available, and the available programming languages. [6][8] This means that any good educational robot should provide as flexible a platform as possible without compromising cost, durability or ease of use. Especially important is that such a robot must come with the necessary sample code and documentation necessary to effectively teach using the platform, and that development (if possible) should be possible using standard SDKs and programming languages. Moreover, if possible, a robot should provide a high-level interface for development so that students do not spend too much time trying to control low-level hardware operation and instead can focus on high-level operation. [6][8]

New Technologies

Numerous technical advances have occurred in the field of robotics and sensors in the past few years, and many of these advancements can be applied to educational robotics. In particular, a series of advancements in sensors and software have made professional production-quality technologies accessible to the point at which they are reasonable and affordable for use in educational robotics. In terms of hardware, the Neato Robotics LIDAR scanner and Microsoft's Kinect RGB-D sensor provide advanced sensing capabilities with both low cost and ease-of-use. [9] These sensors offer capabilities previously only possible through expensive and rare depth cameras, or computationally-intensive stereoscopic vision. These sensors allow for more advanced navigation, mapping and operation than previously expected in an educational robot.

Along with these new sensors, Mecanum wheels, which have long been too rare and expensive for general robotics usage, have become available. These wheels, which allow for

movement in any direction without the need for steering axles, provide maximum vehicle maneuverability with a minimum of drivetrain complexity. [2] While these wheels can be problematic to control, improvements in the processing power of low-power computers make complicated control algorithms feasible. [2] Similarly, these improvements in computer performance (in particular, major performance/Watt power efficiency improvements) mean that real-time image and video processing can be performed on small mobile battery-powered robots using affordable off-the-shelf hardware without unacceptable power consumption.

In addition, these new pieces of hardware are easy to interface with new pieces of software. Willow Garage's ROS is an open-source equivalent to the older MRDS system offered by Microsoft, and it allows the easy abstraction of particular robotic hardware to allow the use of more generic high-level code. Interfacing with new hardware simply requires the availability of new ROS "packages" supporting the hardware, which are then run as distributed services. This produces an extremely flexible platform for robotics development. This abstraction allows for easy integration with robotics education, as development can easily be split up among "nodes" in the ROS package, allowing for easy cooperation and debugging. [1]

A good educational robot must combine flexibility with ease of use and durability. Importantly, given the pace of robotics developments in recent years, it should be easily upgradeable to take advantages of future technologies. Given the expense of investment in educational robotics, it only makes sense for such a platform to be adaptable for years to come. Our project aims to satisfy this through the use of as many off-the-shelf components as possible, including but not limited to sensors, computing hardware and software. Thus, individual components are easy to adapt and replace for different uses. Through the use of standard software like ROS and MRDS, our project allows development in standard programming languages (Python, Java and C++ for ROS; C#, IronPython and others for MRDS) with no need for proprietary SDKs. [1] In particular, by using standard x86/x64 hardware, students can run and simulate their code on their own computers without needed special robotics hardware to test their

code. In addition, the open-source nature of the project allows for easy modification and adaptation by users to continuously improve the platform. [3]

Design Requirements

Previous researchers have attempted to define the ideal features of an educational robot, with Weiss and Overcast identifying the following criteria as essential: platform flexibility and extensibility, quality and durability of hardware, quality of documentation and teaching resources, availability of sample code, continuing developer activity, total system cost, SDKs available, and the available programming languages. [6][8] This means that any good educational robot should provide as flexible a platform as possible without compromising cost, durability or ease of use. Especially important is that such a robot must come with the necessary sample code and documentation necessary to effectively teach using the platform, and that development (if possible) should be possible using standard SDKs and programming languages. Moreover, if possible, a robot should provide a high-level interface for development so that students do not spend too much time trying to control low-level hardware operation and instead can focus on high-level operation. [6][8]

Existing work and the needs of our CS department customers have led to the following requirements:

- The robot must be capable of autonomous indoor navigation and movement, and must be able to easily pass through doorways.
- The robot must be able to use modern high-performance sensors like the Microsoft Kinect in concert with computationally intensive navigation and localization software.
- The robot must have an equivalent level of maneuverability as an adult human (2 degrees of linear freedom, 1 degree of angular freedom, speed of 1.5 m/s) without incurring undue mechanical complexity.
- The robot must be capable of carrying a meaningful payload (~10 kg) and supporting it with the necessary power, data connections, and processing power.

- The robot must be extremely robust; it must be able to survive collisions, protect its electronic equipment from damage, and maintain performance during extended unsupervised operation.
- The robot must provide an accessible software platform that is *both* accessible and powerful, posing limited obstacles to use by novice student users without limiting the capabilities available to more advanced users.
- The robot's software platform must abstract away any and all complexities of low-level hardware control in favor of simple yet powerful high-level control interfaces.
- The robot must be as inexpensive and flexible as possible, using off-the-shelf components whenever possible to limit costs and ease fabrication. In addition, it must be easy to modify without the need of special tools.

Our design provides for these requirements by providing the following characteristics: a flexible and extensible platform, integrated actuators and sensors for easy high-level control, support for production-quality robotics software, and a fully open-source design.

- Flexibility & Extensibility: Our design allows for easy mounting of additional sensors and actuators on the top deck, which can range from simple grippers and cameras to complex models of torsos and faces used for social robotics research. In addition to easy mounting, our design provides easily accessible data and control (USB, Ethernet, Serial, SPI, I^2C, Digital I/O, PWM and Analog input) and power connections (5V & 12V DC).
- Off-the-shelf Structural Elements: Instead of custom-fabricated structural components, our design is entirely comprised of easily available extruded aluminum '80/20' stock with minimal machining. All cutting, drilling and fabrication can be accomplished with hand tools (although it may be easier to fabricate using automatic (CNC) machining), which means that anyone with access to the necessary raw materials can build their own robot.

Joints in the 80/20, while only bolted for simplicity and ease of assembly, are more than strong enough to handle any expected load.

- Mecanum Wheels: Mecanum wheels provide a maximum of maneuverability with a minimum of mechanical complexity in the drivetrain. There are no steering axles, no complicated castering wheel sets and no expensive slip rings for electrical connections. Instead, due to the special design of the wheel, four wheels each equipped with its own motor are capable of movement in any direction and rotation in place. The mechanical simplicity of this design improves reliability of the robot and reduces cost.
- Integrated Actuators & Sensors: Our design provides all the sensors necessary for an inertial measurement unit (IMU)-based navigation system, allow with the sensors necessary for naïve control of the Mecanum-wheel based platform. This allows users to take advantage of the robot's robust sensor and localization capabilities without needing to understand the complexities of the control system in use. In addition, all communication between onboard sensors and control systems will be done over USB connections, making it very easy for students to connect to the robot, and also providing for easy upgrading of the onboard components when necessary.
- Powerful Onboard Computational Resources: The robot is provided with multiple levels of processing power, ranging from a simple microcontroller handling A/D conversion and Digital I/O, to a high-performance mini-PC (equipped with a 2.5 GHz Intel Core i3) for depth and image processing and high-level control. These onboard resources mean that the robot will be able to operate autonomously without requiring off-board processing support. In addition, this allows for high-performance sensors such as the Kinect, which would not otherwise be usable.
- Redundant Control: Control of the robot is provided by a combination of two ARM Cortex-M series microcontrollers, which directly control the motors and handle low-level

sensor data, and an x86/x64 PC that issues high-level commands. As a result, crashes or unexpected behavior in user programs on the PC do not result in a loss of control of the robot or a safety problem. In the case of such a crash or error, the robot will immediately stop, and the user can trigger the start of the "rescue" driver by connecting a USB joystick. Even in the *extremely unlikely* worst-case scenario where all control fails, the individual motor controllers can be programmed to automatically stop the motors and bring the robot to a safe state, or the user can trigger the emergency stop switch on the top of the robot.

- Production-Quality Software Support: We intend for our robot to be fully supported in Willow Garage's ROS and possibly Microsoft's MRDS in addition. ROS and MRDS are two of the most used robotics platforms in commercial and research use, and full support for ROS and MRDS makes it easy to integrate the robot into existing teaching and research projects which are already using these software suites. Importantly, this lets our robot take advantages of the robust high-level functionality (search, path planning, mapping, etc.) provided in these systems.
- Complete Drive System Abstraction: To compensate for any inherent difficulties in the use of Mecanum wheels, our software completely abstracts away the complexity of the drive system. Users need not understand the subtleties of the implementation; instead, they simply provide a linear and angular velocity vector (effectively standardized in ROS as Twist message, better known as the "cmd_vel" topic, which provides <x,y,z> linear and <x,y,z> rotation commands) which is then effected by the drive algorithms. A combination of the inertial measurement unit, wheel rotation encoders, and external references can be used to ensure that the robot either completes the commanded movement or alerts the user of the failure to do so.
- Familiar Programming: ROS in particular supports programming in Python, Java and C++. Almost all students using the robot will have previous experience with at least one

of these languages, which improves the accessibility of the robot to new users. This allows for easy integration with existing curricula without requiring that students be taught new programming languages.

- Fully Open Source Design: Our design will be one of the exceedingly few open source robotics platforms available, and it will be the only medium-sized non-humanoid open source robot available. Open source means CAD design files, circuit designs, program source code, bill of materials, documentation and a detailed reference implementation are all available under a suitable license (in this case, a modified BSD license). This project must be well enough documented so that potential users will be able to quickly and easily fabricate and assemble their own robot while also having the ability to easily modify the design to suit their own particular requirements. As part of this open source design, modifications and improvements by other end users can be contributed back to the project and used to continually improve the design for all users.

Design Alternatives

Existing Literature

Very little literature exists on the topic of educational robotics, and the small amount that does is limited almost exclusively to the topic of using robots in education. The overwhelming consensus from the literature is that robots provide an ideal hands-on environment for teaching computer science concepts, however, there is little focus on the robots themselves. [6][7][8] The little available work that focuses on the robotics platforms themselves is entirely focused on standard commercial platforms such as LEGO NXT or Parallax Scribbler. [6][8] The previously mentioned work of Overcast and Weiss is unique in that it discusses the necessary characteristics of educational robots that we have incorporated into our design requirements. [6] They are:

- Platform flexibility and extensibility
- Quality and durability of hardware
- Quality of documentation and teaching resources
- Availability of sample code
- Continuing developer activity
- Total system cost, SDKs available
- Available programming languages

Existing Designs

A variety of open source robotics projects exist, however, they are usually limited to specific pieces of hardware (arms, grippers, sensors, etc.) designed for use on commercial platforms. However, there is virtually no existing work on open source educational robotics *platform* development aside from CMU's legged 'Chiara'. [10] Chiara has impressive maneuverability but lacks powerful onboard processing and payload capability. In contrast, 'affordable' available commercial platforms like Mobile Robot's Pioneer series can be equipped

with powerful computers but are extremely simple robots (standard differential drive) that trade maneuverability and features for ruggedness. Simpler still is the iRobot Create, widely used for its payload flexibility despite its few sensors and limited performance. In short, there are no affordable robots that combine payload flexibility and innovative features such as higher maneuverability.

Alternatives

The distinct lack of existing designs means that our project is driven almost entirely by the requirements of the customers: flexibility, maneuverability, robustness and ease-of-use. As expected, there are a variety of means of implementing each of these, and a variety of alternatives were examined for the core systems of the robot. In particular, the areas examined were: drive system, sensor suite, and software stack.

1. Drive System

   As human level maneuverability is a basic design requirement, drive systems were limited to those systems capable of holonomic motion:

   a. Omni wheels
      i. Simple, cheap, widely used
      ii. Mechanically simple
      iii. Low performance, hard to find at larger sizes
      iv. Complicated control
   b. Mecanum wheels
      i. Available for larger robots (≥6" diameter)
      ii. Complicated control
      iii. Mechanically simple
   c. Omni-directional steering
      i. High performance

ii. Comparatively simple control

iii. Mechanically complex

iv. Expensive to implement

Omni-directional steering was quickly eliminated as an option due to the expense and complexity of building multiple steering drive wheels. Such complexity not only increases costs, but also adds new components to fail. In particular, as the drive system of the robot is likely the primary source of hardware failures (there being no other major *mechanical* system onboard), it is in our interest to use as simple a mechanical design as possible to improve reliability. The choice between Omni wheels and Mecanum wheels was made based on the commercial availability of 6" Mecanum wheels with sufficient load capacity for use on our robot and the lack thereof of similarly suitable Omni wheels.

In addition, the chassis designed for use with Mecanum wheels could, in the event it was necessary, be fitted with conventional tires or tracks and continue to function (albeit without the holonomic drive capability), which could not be done with a chassis designed for Omni wheels. While this compatibility will likely never be used at Union, it provides a method for modifying the design for use on unprepared surfaces (off road, snow, etc.) for which Mecanum wheels are less suitable. This design flexibility is an important part of designing the robot to be used and improved by multiple users.

2. Sensor Suite

As noted in the customers' requirements, the robot must be capable of self-contained indoor autonomous navigation. Only a few technologies are usable for this:

a. Dead reckoning
    i. Simple
    ii. Cheap
    iii. Poor accuracy
    iv. Requires no external data
b. Inertial Measurement
    i. Complex software
    ii. Fairly accurate
    iii. Requires no external data
    iv. Requires special sensors
c. Ultrasonic sensors
    i. Cheap
    ii. Easy to use
    iii. Poor accuracy
d. Infrared sensors
    i. Cheap
    ii. Easy to use
    iii. Poor accuracy
    iv. Limited range
e. LIDAR
    i. Accurate
    ii. Long range
    iii. Easy to use

iv. Extremely expensive

f. Microsoft Kinect

i. Medium range

ii. Reasonably accurate

iii. Requires a fairly powerful computer

iv. Reasonably affordable ($150 US)

Ultrasonic and Infrared sensors were, quickly eliminated due to poor performance. Given the size and speed of the robot, more accuracy is required than these sensors could easily provide. LIDAR, while ideal for indoor navigation tasks, was ruled out due to costs. The simplest LIDAR unit on the market is available for $400 US as a part in a robotic vacuum; the cheapest standalone LIDARs retail above $2000 US. As minimizing total cost is an important part of the project, it is simply unacceptable to use a sensor that costs as much as the entire rest of the robot.

By process of elimination, our sensor suite is based on the Microsoft Kinect and an Inertial Measurement Unit (IMU). The IMU is necessary for gross navigational tasks, while the Kinect is necessary for precise tasks, like handling doorways, localization to update the IMU, and object avoidance. Of note is that both systems are reliant on substantial software processing of data; like the choice of Mecanum wheels, we have accepted increased software complexity to improve other aspects of the design.

3. Software Stack

Few software stacks exist for robotics programming, and fewer still are easy to extend to support a completely new robot:

a. Robotic Operating System (ROS)
    i. Open source
    ii. Code in Python, Java, C++
    iii. Recently released
    iv. Provides concurrency & communications primitives
    v. Linux/UNIX only

b. Microsoft Robotics Developer Studio (MRDS)
    i. Proprietary
    ii. Fairly widely used
    iii. Code in C#, VB.net, C++, other CLR languages
    iv. Provides concurrency & communications primitives
    v. Windows only

Neither platform has clear technical superiority, although MRDS has the benefit of extensive use in major robotics projects (DARPA Grand Challenge, among others) and established user base. However, the Windows-only compatibility and closed-source nature of MRDS, along with growing user base of ROS make ROS the better choice for this project. Ideally, if time permits, we also intend provide support for MRDS, as broader compatibility makes it easier for other users to use our design.

After considering all available alternatives, our design consists of Mecanum wheels for the drive system, Microsoft Kinect and Inertial Measurement Unit for navigation, and the Robotic Operating System running on Ubuntu Linux computers for the software stack. All of these choices are the result of maximizing performance while minimizing monetary costs; this means

other costs, like the computational complexity of some operations, are increased. This is particularly true for the drive and sensor systems; both were explicitly chosen to improve performance at the cost of increased software complexity. This is an acceptable choice for this project because software can be comparatively easily rebuilt or replaced whereas complex, hard-to-maintain hardware is often abandoned or broken. By keeping mechanical complexity and costs down, we make it easier for others to build and maintain their own robots and we increase the reliability of the design.

Component Overview

- **Onboard Computer:** Necessary to provide high-level control of the robot, video and image processing, and SLAM (Simultaneous Localization And Mapping) using the Kinect sensor and ultrasonic sensors. Additionally provides the computational resources for Human-Computer Interaction (HCI) payloads. The computer chosen and purchased is an ASRock CoreHT 231D with a 'Sandy Bridge' Intel Core i3 2310M processor.
- **Mbed Microcontrollers:** Controls the robot's motors and receives sensor information from all sensors except for the Kinect. One microcontroller (an ARM Cortex-M3 design) handles communication with the serial motor controllers and processes the outputs of the Hall-effect quadrature encoders attached to the motors to provide speed feedback information. The other microcontroller (an ARM Cortex-M0 design) reads data from the analog accelerometer, analog gyroscope, and the eight bumper impact sensors.
- **Kinect:** Extremely low cost color and depth sensor necessary for *affordably* implementing Simultaneous Localization And Mapping (SLAM). Kinect offers similar performance to LIDAR sensors at an order of magnitude lower cost ($150 versus $4000).
- **Accelerometer, Gyroscope & Impact Sensors:** Sensors necessary to provide accurate motion control, dead-reckoning navigation, collision avoidance and allow the robot to operate safely and successfully in an indoor environment.
- **Mecanum Wheels:** Necessary to provide the *mechanically simplest* drive system with human-level mobility which is necessary for HCI research work. Human-level mobility means the ability to drive in all directions (forwards, backwards, left and right) without turning, and the ability to turn in place. Mecanum wheels allow such motion with the minimum of moving parts, which provides for a more reliable and robust robot.
- **Motors, Motor Mounts & Motor Controllers:** Used to control the motors and move the robot. Due to the design of the Mecanum wheels, one motor is needed per wheel. The

selected motor mounts allow easy attachment of the motors and wheels to the 80/20 structural frame.

- **Batteries, Charger & Power Converter:** Provides reliable and constant power to all electrical system on the robot, ranging from the drive system to the sensors. The use of multiple batteries allows longer runtime, while the use of the DC-DC power converter provides the stable voltage needed by the onboard computers to operate. Additionally, the onboard 15V DC power supply provides a self-contained charging system, which allows future users to implement an automatic charging system without need for an external charging station.
- **80/20 Aluminum, Aluminum Structure, Fasteners & Polycarbonate Sheet:** Structural components for building the robot. The use of standard 80/20 components provides for easy assembly and disassembly, repair, and modifications. This flexibility is important for a robot designed to be used for research and teaching over multiple years.

Final Design

In contrast to our early experimental designs, our last major design had the wheels situated underneath the robot with the axle supported on both sides. The vertical components of the body stretched downwards to form the outside support of each while an additional support was added in on the inside of the robot for the other side of the axle. The motors would be mounted on a horizontal support member connected to the two vertical pieces. This would eliminate the stress on the motor bracket and motor as well as reduce the stress on the axle. Bearings for the axle would be inserted inside the 80/20.

In addition, a bumper system was added. This would alleviate the delicate nature off the electronics, giving the robot additional survivability. In addition, the dimensions of the body of the robot were reduced to 18” by 18” by 8”, in comparison to earlier dimensions of 24” by 24” by 12”. This, coupled with a 3” buffer on each side of, brings the ground footprint of the robot to 24” by 24”. The batteries would be situated side-by-side in the rear of the robot, with the accelerometer situated in the very center and the computers in the front.

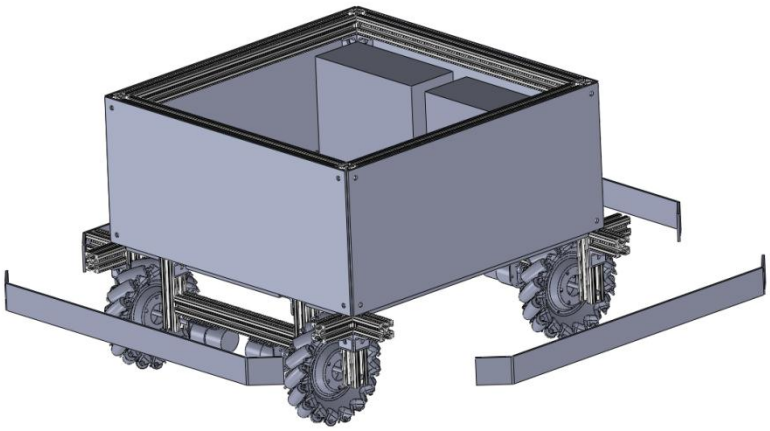

Figure 1 - 3rd design, looking at front left quarter. Note the two batteries in the rear and the new structure of the bumpers.

Even given the newly shrunk dimensions of the robot, it is still a little too big to easily fit through doorways. In addition, the batteries, which are the heaviest part of the electronics, would

cause the pressure on the rear wheels to be greater than the front wheels. This would cause difficulties when the robot was attempting to drive sideways, as a difference in pressure could lead to a difference in thrust between the two sets of wheels, causing the robot to twist unexpectedly. While not necessarily a problem with a conventional drivetrain, existing work with Mecanum wheels demonstrated better performance when weight was evenly distributed among all four wheels.

The final revised design we settled on for the Advanced Educational Robot consists of a body with dimensions 18” by 18” by 6” constructed out of 80/20 and aluminum plating with Mecanum wheels below and underneath the body supported by 80/20 on either side. The design also includes bumpers set 3 inches in front and behind the body of the robot and 1.5 inches to either side of the robot. The batteries are mounted below the body in the middle of each side.

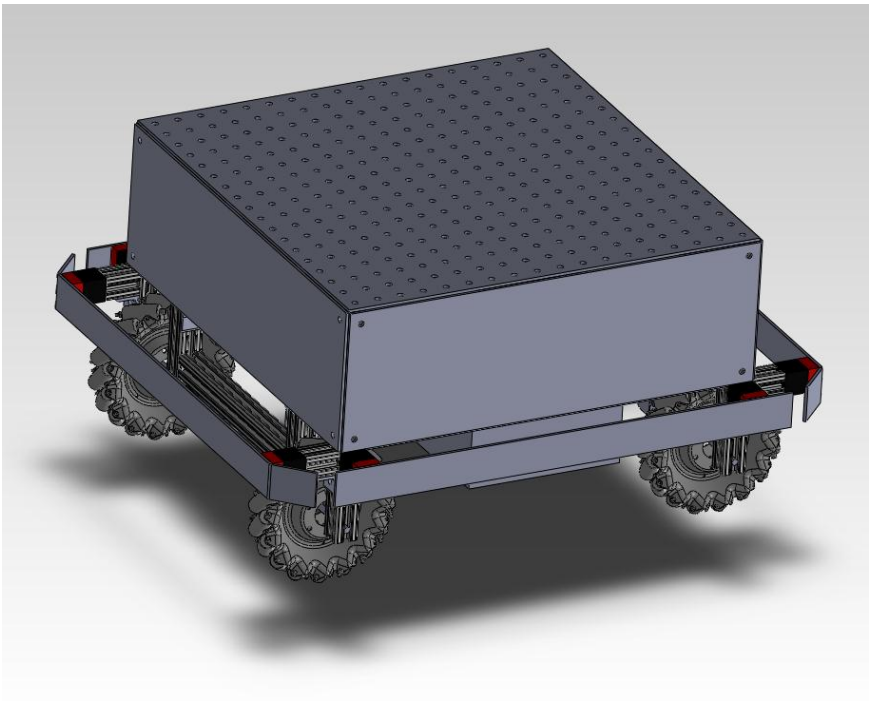

Figure 2 - Rendering of final design, exterior sensors not shown. Note batteries underneath the body.

As previously described, the axles for the wheels are mounted to both the main vertical corner pieces and to short vertical members on the inside, as shown in Figure 3 below.

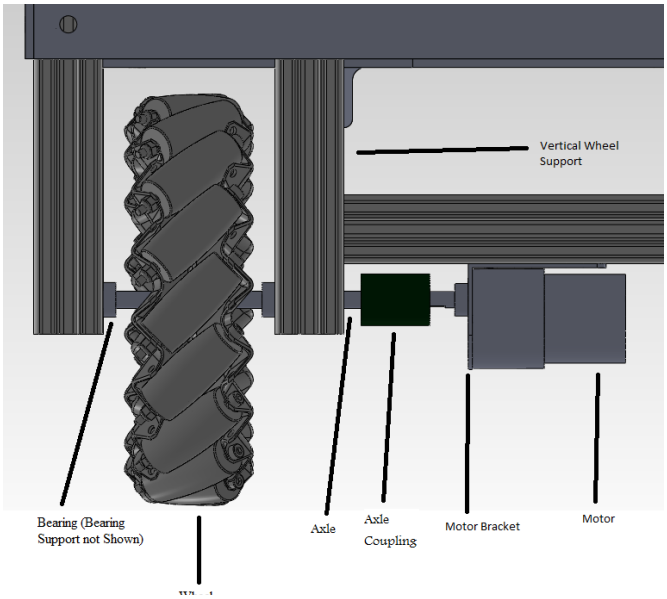

Figure 3 - Annotated diagram of the drive system for each wheel.

Similarly, the bumper system is the same as described previously, comprised of the aluminum bumper itself, shock absorbing blocks, and touch sensors. Shown below in Figure 4 is a corner of the robot, illustrating the differences between the front/rear bumpers and the side bumpers.

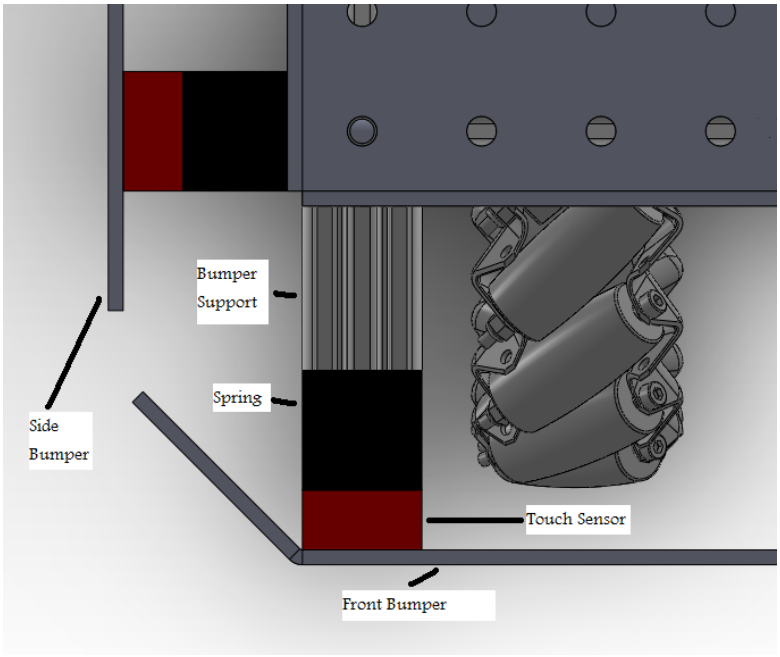

Figure 4 - Front and side bumper layout.

The final assembled robot is shown below in Figure 5. Note the Kinect sensor atop the robot and the red emergency stop button at the rear.

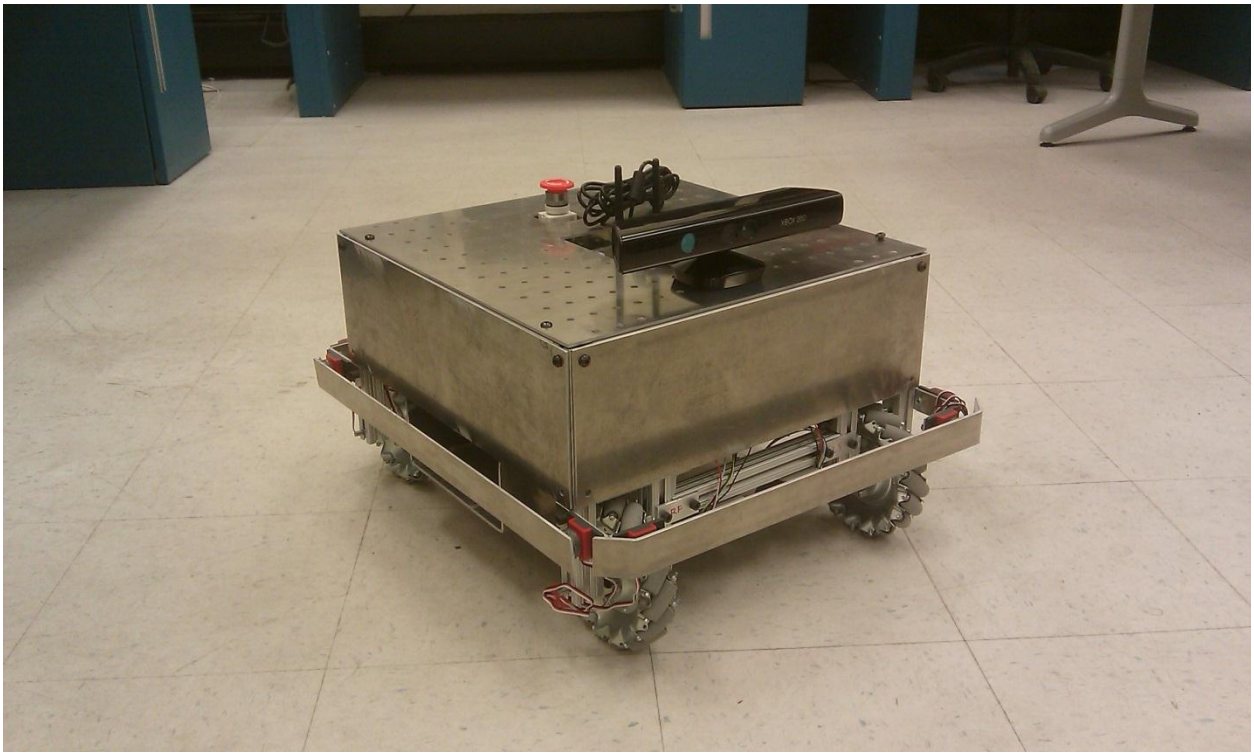

Figure 5 – Completed Robot

Design Details

Motors

In order to achieve the desired maximum velocity in the desired time (1.5 m/s in 3 seconds), the robot must accelerate at a rate of 0.5 m/s^2. The predicted maximum payload of the robot is 40 kg. Using the equation Force = mass*acceleration, we find that the motors must provide a total of 0.5*40 = 20 N of force. As there will be 4 motors, each motor must produce 5 N of force. The wheels will have a radius of 3" (0.0768 m). Thus, the required torque = force*distance = 5* 0.0768 = 0.3846 N*m. The Pololu 37D mm Metal Gearmotor with a gear ratio of 67:1 has a listed stall torque of 200 oz-in. This is equal to 1.412 N*m. Thus, the stall torque of the motor is 3.76 times the desired torque, ensuring that the chosen motor can achieve the required acceleration.

Axles

The axles would bear the weight of the entire robot during its normal operations and also need to be strong enough to maintain structural integrity in the event the robot was dropped into its wheels. To confirm that the chosen axles could support the weight of the robot, we reduced the axle system to a beam supported on both sides with a point load in the center. This is a simplification of the actual system, but it is useful for a conservative estimate. The robot will weight around 100 lbs. which under normal circumstances will be spread evenly on each of the 4 axles, meaning that each one must support 25 lbs.

Using the equation for the maximum moment in a simply supported beam with a load in the center M_{max} = P/2*L/2 where P is the load and L is the distance between the two supports. In this instance, the distance between the two axle supports is 3 inches, meaning that M_{max} = 25/2*3/2 = 18.75 lb*in. Using the equation for maximum stress caused by a applied moment σ_m= M*c/I where M is the moment, c is the distance away from the neutral axis, and I is the moment of inertia of the beam. In this case where the beam is a circular, I = 1/4*pi*r^4 and c = r where r is

the radius of the beam cross-section (0.125”). Thus $\sigma_m = 18.75/(.25*pi*.125^3) = 12.2$ ksi. The steel used in the axle (1035) has a yield strength of 70 ksi, giving the axles a factor of safety of 5.7. This shows that the axles chosen have enough strength to endure any but the most sudden of drops.

Bearings

The bearings were chosen from McMaster-Carr with the requirement that the inner diameter be the same as the axle diameter and that the outer diameter be small enough to fit in a support bracket without reducing its structural integrity. In addition, it was made to be double shielded to not require maintenance and lubrication. Each of these bearings has a Dynamic Radial Load Capacity of 243 lbs., giving a single bearing a factor of safety of 2.4 for the entire robot, so as a whole each bearing will have a FOS of 19.44. This ensures that the bearings will not fail under the expected operating conditions.

Use of 80/20

We chose 80/20 because of its availability, flexibility, and ease of use. 80/20 consists of T-Slotted Extruded Aluminum beams which allow for the attachment of bolts at any point along all four of their faces. This allows us to construct the frame of the robot with a minimum of machining, allowing institutions without the same caliber of machine shop as Union to construct it as well. In addition, the slotted nature of the 80/20 allows for the inclusion of additional attachments after the design and construction process has been completed, increasing the adaptability of the final product to suit the needs of our customers.

Plating Material

We chose the material for the top and bottom places to be aluminum because the bottom plate needed to be strong to support the batteries and the computer and the top plate needed to be

strong in order to accommodate the possible attachments our customers wanted. For instance, in order to be used for human-robot interaction, a torso would need to be mounted on the robot that would stretch to approaching human height. This would have to be steady, and a light, strong metal was the best choice for the mounting plate.

Though the side plates do not serve any structural purpose, we decided to keep them as aluminum for the sake of simplicity. It reduces the number and variety of metals we would need to deal with, and the additional strength aluminum siding would provide would help protect the electronic components in case of a collision with an object that missed the bumper. In addition, the use of aluminum only adds approximately 2 pounds to the final weight of the robot, which is a minor increase in weight compared to the fully loaded weight of almost 100 pounds.

Control System

An overall diagram of the control system of the robot is shown below in Figure 6. As one can see, the control system uses the two microcontrollers to handle all communication with the drive system and the vehicle's sensors. Additionally, one of the two DC-DC converters is connected to the control computer to provide information about the robot's battery voltage.

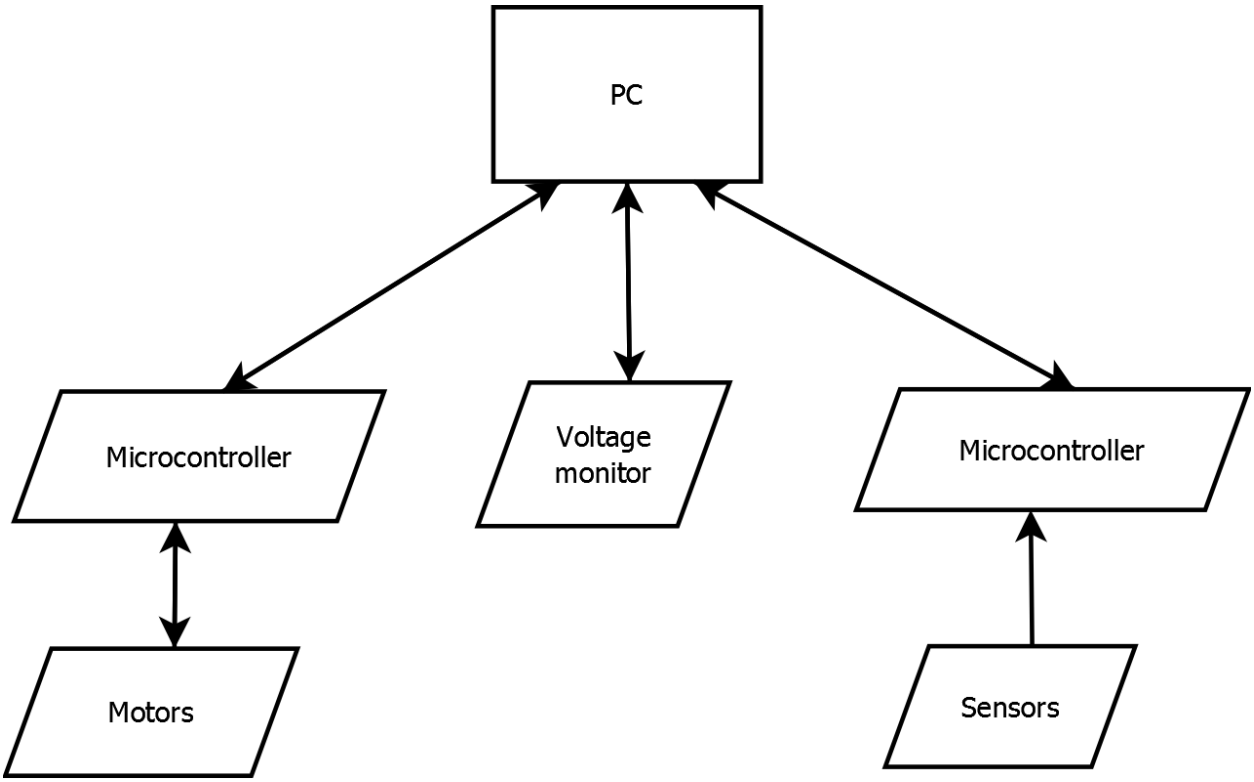

Figure 6 – Control System

This control architecture provides nearly complete abstraction of the hardware implementation details of the robot; the microcontroller that handles motor control provides a standard interface that masks the complexity of communication with the motor controllers or the encoders, while the microcontroller that handles communication with the sensors does the necessary A/D conversion and control of the analog and digital sensors connected to it.

Maintaining this level of abstraction is important to the robot's design because it lets users dramatically change certain parts of the robot without having to replace the entire control system. For example, future users who may design and build a suspension system may find that they need an entirely different motor control system. Since that system is completely abstracted away from the control PC and the software running on it, there will be no need for them to change the existing driver software.

We have included the full code for both microcontrollers in Appendix B.

ROS Driver

To illustrate the power of the control system's hardware abstraction, we have included the entire code of the three main ROS nodes that make up the ROS driver in Appendix A. ROS is a node-based system that uses TCP and UDP to handle communication between nodes from different packages. ROS provides basic tools to build asynchronous and distributed software without the complexity of directly manipulating threads and inter-process communication.

The ROS driver for our robot is extremely simple yet powerful. It consists of three main nodes, with one node handling motor control, one node handling sensor data, and one node handling the voltage monitoring. This structure is shown below in the diagram produced by the built-in ROS tool **rxgraph**, Figure 7.

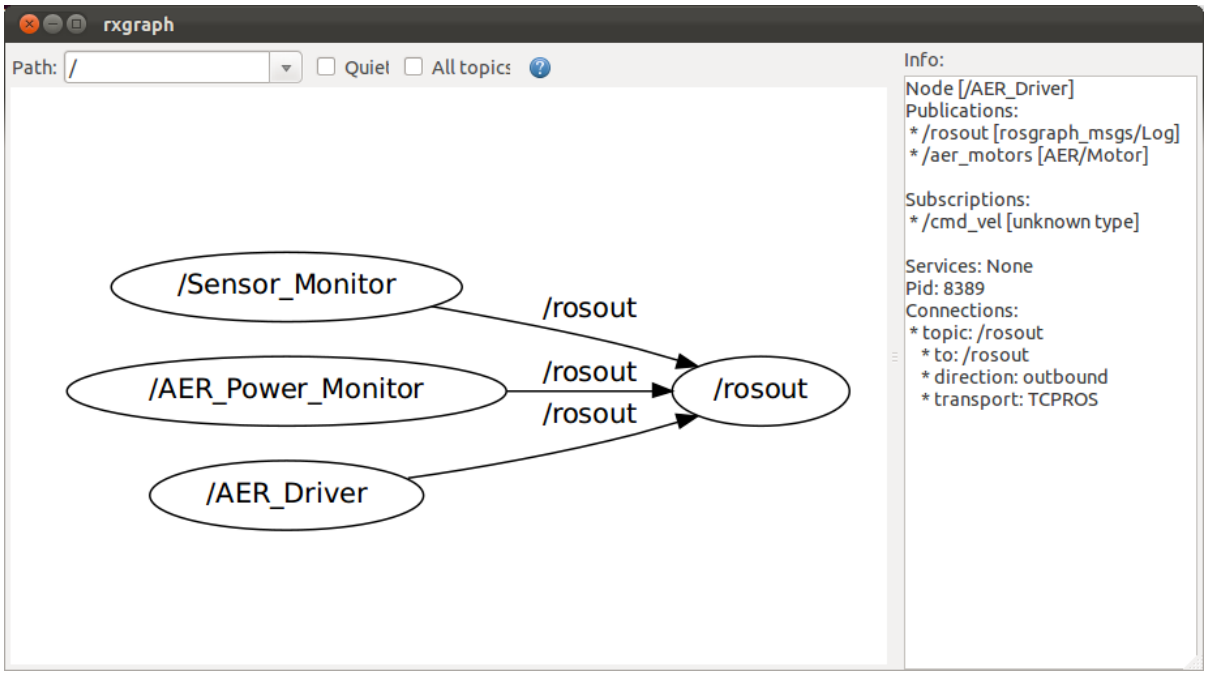

Figure 7 – ROS Driver Nodes

The code for each of these nodes can be found in Appendix A. In the interest of space, we have not included in Appendix A the Python classes that directly handle communication with the

microcontrollers or the Python class that provides the Mecanum wheel drive algorithm. These classes continue the system of abstraction used with the microcontrollers by providing a simple consistent object interface to the ROS nodes. Thus, if a future user wished to completely redesign the underlying control implementation, they could reuse the existing ROS driver nodes provided they maintained the current object interfaces.

In addition to the core nodes of the ROS driver, we have also provided a Teleoperation node designed to control the robot using a USB game controller (currently tuned to an Xbox 360 controller, but the button and axis mappings can be easily changed), a Kinect navigation node that provides basic hallway-following navigation, and number of visualization tools that let users see the outputs of the robot's sensors and the commands being sent to the robot's motors. Additionally, we have provided a number of shell scripts to quickly run a variety of driver configurations.

Performance

The robot largely meets or exceeds the design requirements set forth. (See the Design Requirements section for more details) In terms of vehicle performance and maneuverability, it exceeds the 1.5 m/s forwards velocity goal and provides all the expected maneuverability of a vehicle with holonomic drive. However, the drive system is very sensitive to irregularities in flooring and changes in floor material. These changes and irregularity almost always result in wheel slippage, which causes severe performance problems. Most importantly, wheel slippage results in unexpected rotation or sideways drift that is difficult to correct for.

We have found that the onboard sensors are not accurate enough to provide data for an effective closed-loop control system that would mitigate these problems; in particular, while we can *detect* wheel slippage fairly easily using the encoders, we cannot *correct* for this wheel slippage. This is largely because there is no clear answer as to the proper correction for wheel slippage; increasing speed is likely to increase wheel slippage without increasing traction, while decreasing speed is likely to increase the deviation from the expected direction of movement.

Production Schedule

The construction of the robot consisted of four main phases: hardware design, hardware construction, software design, and software implementation.

Hardware design of the entire vehicle was largely completed during the fall term so that all parts could be fabricated by the early part of winter term. This phase should have been completed earlier in the term so that all parts could be completely fabricated and the robot assembled before the beginning of winter term. This would have provided more time for software and hardware testing. Hardware construction was largely completed in the first half of winter term, with some minor parts, like the bumpers and top plate, being assembled near the end of winter term. As mentioned already, this should have been completed in the fall term to provide more time for testing.

Software design proceeded concurrently with the hardware assembly in the first half of winter term, followed by the implementation of the ROS driver, which was complete shortly after the sixth week of the term. These phases of the project were significantly faster than expected, which suggests that they could have been accomplished earlier in the project. This would have been possible due to the system of abstraction in the control system that lets major parts be designed and implemented before the completion of other parts.

Due to the longer-than-expected hardware construction phase of the robot, less time was available for testing than we had expected. In light of the previous suggested changes to the schedule, completing them major phases of construction earlier would have provided the necessary time to complete more rigorous testing than we were able to complete in the final weeks of winter term. This increased testing would have allowed us to better characterize the problem of wheel slippage and evaluate methods of mitigating it.

Cost Analysis

Our project has been completed under budget and significantly under the cost of competing commercial equivalents. Our design requires no special tools to assemble, and requires relatively little complex component fabrication, which helps reduce the effective cost to an end user. Similarly, the actual assembly stage of the fabricated parts is fairly short, with assembly being easily completed in a day. This too contributes to a low effective cost for end users. In fact, our design can be easily fabricated with the resources available to any standard undergraduate or graduate machine shop, and can be assembled by users with no particular experience in building robots.

Equivalent commercially available robotics platforms, like Mobile Robot's Pioneer 3 series, cost well over $1000 US more than our robot in the most basic configuration, and well over $3000 US more when equipped with comparable computers and sensors. This increase in cost does not come hand in hand with better performance; our robot offers better maneuverability, greater size, and significantly enhanced expansion options. In addition, our ROS driver and software support is significantly easier to use than either of the existing ROS drivers for the Pioneer 3 series.

Future users or those who want to build their own version of our robot can reduce the cost of construction by fabricating more of the materials for the robot themselves instead of purchasing commercially available equivalents. In particular, the angle brackets used to connect the 80/20 frame could easily be replaced with brackets made from angle stock aluminum. Similarly, users with particular computing needs can build their own computers instead of purchasing complete systems (any small Mini-ITX form-factor computer should fit the chassis) to reduce cost and improve performance.

For those users constrained by fabrication and labor costs, the most complex parts of fabrication, namely the bent bottom plate and the bent motor brackets, can be replaced with simpler multi-part designs with only a slight increase in weight. Likewise, users with limited

access to water-jet cutters or similar equipment can replace many of the large aluminum plates of our design with equivalent components fabricated from acrylic or polycarbonate plastic. Acrylic plastic, in particular, can be cut using commonly available laser-cutters. As few of these plates are structural, they can be replaced with lighter and cheaper material without reducing performance. As already noted, the actual assembly of the robot requires no specialized tools, and so users could have all parts fabricated externally and then assemble their robot onsite with a minimum of equipment.

A complete budget for the robot can be found in Appendix C, note that while we initially purchased a Pandaboard ARM Single-Board-Computer, we chose not to use it in our control system so this cost is not represented in the complete budget for the robot.

Conclusions

We have successfully designed and built an open-source robotics platform for education and research use that provides greater functionality at a lower cost than competing commercial alternatives. Our platform provides greater maneuverability, expansion capability, and onboard processing power than commercial systems, yet can be easily fabricated using the resources available to any undergraduate or graduate institution's machine shop. In particular, we have used off-the-shelf components wherever possible to minimize the need for component fabrication and ease assembly. As already noted, the cost of our robot could be reduced by fabricating more of the parts specifically, which future users may find to be more cost-effective.

Our robot largely meets or exceeds the design requirements:

- The robot must be capable of autonomous indoor navigation and movement, and must be able to easily pass through doorways. ***Met*** – Our robot, through its robust ROS driver and high maneuverability, can be easily controlled both directly and autonomously to navigate in indoor environments including a wide variety of floor conditions, doorways, and hallways.
- The robot must be able to use modern high-performance sensors like the Microsoft Kinect in concert with computationally intensive navigation and localization software. ***Met*** – Our robot provides plentiful processing power, dedicated ports for a Kinect sensor, and runs the necessary software drivers for navigation using these high-performance sensors.
- The robot must have an equivalent level of maneuverability as an adult human (2 degrees of linear freedom, 1 degree of angular freedom, speed of 1.5 m/s) without incurring undue mechanical complexity. ***Exceeded*** – Our robot provides >1.5 m/s linear maneuverability in all directions on ideal flooring and >1.5 m/s forwards and backwards

on less-than-ideal flooring environments with reduced traction. Our robot is easily capable of matching the maneuverability of an adult human in an indoor environment.

- The robot must be capable of carrying a meaningful payload (~10 kg) and supporting it with the necessary power, data connections, and processing power. ***Met*** – Our robot provides all necessary power and data connections (USB, Kinect, 12V DC) to user payloads and additionally provides the necessary emergency stop control to these payloads.
- The robot must be extremely robust; it must be able to survive collisions, protect its electronic equipment from damage, and maintain performance during extended unsupervised operation. ***Exceeded*** – By virtue of its aluminum frame and body, our robot is extremely resistant to damage from impacts. This has been verified in testing through repeated test collisions with the testing enclosure. Through several levels of shock isolating foam and mountings, our robot protect the onboard electronics from damage due to impacts or drops.
- The robot must provide an accessible software platform that is *both* accessible and powerful, posing limited obstacles to use by novice student users without limiting the capabilities available to more advanced users. ***Exceeded*** – The software for our robot provides simple yet powerful tools to novice and experienced users, both through ROS using the provided ROS driver, and through their own code using the provided Python classes (which are used by the ROS driver).
- The robot's software platform must abstract away any and all complexities of low-level hardware control in favor of simple yet powerful high-level control interfaces. ***Exceeded*** – The control system and software for our robot maintains simple and consistent levels of abstraction that completely separate the complexities of hardware and software implementation at every level, allowing easy modification and replacement of every stage

of the control system. Through the use of standard ROS ‘best practices’ we have made using our robot at a high level *easier* than that of existing commercial robots.

- The robot must be as inexpensive and flexible as possible, using off-the-shelf components whenever possible to limit costs and ease fabrication. In addition, it must be easy to modify without the need of special tools. ***Exceeded*** – Our robot has been complete under budget and is significantly more cost-effective than commercial platforms. Our robot is easy to assemble and requires no tools for regular maintenance other that standard screwdrivers.

Future Work

In light of problems we encountered with wheel slippage and traction, future users are strongly recommended to evaluate the possibility of designing and implementing a suspension system. Adding a suspension system to the robot will ensure that all wheels maintain constant contact with the ground, which should dramatically reduce the occurrence of wheel slippage. As wheel slip is currently the primary cause for performance problems, reducing wheel slippage will allow for significant improvements to vehicle performance. Reducing wheel slip will also improve vehicle performance on different flooring surfaces and conditions, something that has been a problem in some of our tests. Future users are advised to consider the driving surfaces and traction available to their robot before using our design, since mecanum wheels are not necessarily appropriate for all conditions and environments. For users who are interested in operating our robot in an outdoor environment, we strongly suggest replacing the mecanum wheels with standard pneumatic tires and using the alternative drive control algorithms we have provided specifically for this purpose.

In addition to drive train improvements, future work on our robot can include the development of an accurate indoor navigation system. The accelerometer and gyroscope we used on the robot are not accurate enough or resistant enough to vehicle vibration to provide for an accurate navigation system. However, development of an indoor navigation system using the Kinect sensor and other methods of tracking external references is certainly possible and would improve vehicle usability. In particular, a system capable of accurate navigation in constrained hallway spaces and classrooms would be ideal, as such performance would allow for extended autonomous operation in human environments without continuous human supervision. Achieving this level of performance would allow the robot to be used for extended human-machine-interaction studies while critically providing for the safety of both the robot and human participants.

User's Manual

The user's manual is included in the ROS Package, both as a PDF "Manual.pdf" and as a raw text file "README.txt"

Appendix A – ROS Driver Code (Selected Components)

AER_Driver.py

```
#!/usr/bin/python

from PlatformComponents import *
from DriveControllers import *
from TestComponents import *

import roslib; roslib.load_manifest('AER')
import rospy

from AER.msg import Motor
from geometry_msgs.msg import Twist
from std_msgs.msg import String


class AERdriver:
    def __init__(self, real):
        rospy.init_node('AER_Driver')
        if real:
            port = rospy.get_param('AER_Driver/control_port')
            self.robot = AERplatform(port, 115200, 'platform')
        else:
            self.robot = EMUplatform("/dev/emulator", 230400, 'emulator')

        self.controller = MecanumDrive()

        rospy.Subscriber("cmd_vel", Twist, self.callback)
        self.state_pub = rospy.Publisher("aer_motors", Motor)
        rate = rospy.Rate(rospy.get_param('~hz', 60))

        while not rospy.is_shutdown():
            rate.sleep()
            latest_state = self.robot.GetState()
            motor_message = Motor()
            motor_message.front.M0 = latest_state.LF
            motor_message.front.M1 = latest_state.RF
            motor_message.rear.M0 = latest_state.LR
            motor_message.rear.M1 = latest_state.RR
            self.state_pub.publish(motor_message)

    def callback(self, data):
        """Send Twist commands to the robot"""
        X = data.linear.x
        Y = data.linear.y
        Z = data.angular.z
        #print "X : " + str(X) + " Y : " + str(Y) + " Z : " + str(Z)
        commands = self.controller.Compute(X, Y, Z)
        self.robot.Go(commands[0], commands[1], commands[2], commands[3])

if __name__ == "__main__":
    AERdriver(True)
```

Power_Monitor.py

```
#!/usr/bin/python

from DCDC_USB_Monitor import *

import roslib; roslib.load_manifest('AER')
import rospy

from AER.msg import Power
```

```
class PowerMonitor():

    def __init__(self):
        rospy.init_node('AER_Power_Monitor')
        self.dcdc = DCDC_USB()
        self.power_pub = rospy.Publisher("aer_power", Power)
        rate = rospy.Rate(rospy.get_param('~hz', 1))

        while not rospy.is_shutdown():
            rate.sleep()
            power_message = Power()
            power_dict = self.dcdc.GetVoltages()
            power_message.input = power_dict["input"]
            power_message.ignition = power_dict["ignition"]
            power_message.output = power_dict["output"]
            self.power_pub.publish(power_message)

if __name__ == "__main__":
    PowerMonitor()
```

Sensor_Monitor.py

```
#!/usr/bin/python

from PlatformComponents import *

import roslib; roslib.load_manifest('AER')
import rospy

from AER.msg import Sensor
from AER.msg import Bumper


class SensorMonitor():

    def __init__(self):
        rospy.init_node('Sensor_Monitor')
        port = rospy.get_param('Sensor_Monitor/sensor_port')
        self.micro = AERsensor(port, 115200, "Sensor platform")
        self.bumper_pub = rospy.Publisher("aer_bumpers", Bumper)
        self.sensor_pub = rospy.Publisher("aer_sensors", Sensor)
        rate = rospy.Rate(rospy.get_param('~hz', 60))

        while not rospy.is_shutdown():
            rate.sleep()
            sensor_message = Sensor()
            bumper_message = Bumper()
            sensor_state = self.micro.GetState()
            bumper_message.bumper1 = sensor_state[1].RFF
            bumper_message.bumper2 = sensor_state[1].RFS
            bumper_message.bumper3 = sensor_state[1].RRS
            bumper_message.bumper4 = sensor_state[1].RRB
            bumper_message.bumper5 = sensor_state[1].LRB
            bumper_message.bumper6 = sensor_state[1].LRS
            bumper_message.bumper7 = sensor_state[1].LFS
            bumper_message.bumper8 = sensor_state[1].LFF
            sensor_message.accelerometer.x = sensor_state[0].X_A
            sensor_message.accelerometer.y = sensor_state[0].Y_A
            sensor_message.accelerometer.z = sensor_state[0].Z_A
            sensor_message.gyroscope.z = sensor_state[0].Z_R
            self.bumper_pub.publish(bumper_message)
            self.sensor_pub.publish(sensor_message)

if __name__ == "__main__":
    SensorMonitor()
```

Appendix B – Microcontroller Code

Motor Control Microcontroller (ARM Cortex-M3)

```
#include "mbed.h"
#include "QEI.h"

Serial pc(USBTX, USBRX);
Serial qik1(p9, p10);
Serial qik2(p13, p14);

DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
DigitalOut led4(LED4);

float LF_speed = 0.0;
float RF_speed = 0.0;
float LR_speed = 0.0;
float RR_speed = 0.0;

char commands[8];

Timeout autostop;
Ticker execute;
// Wheel encoders
QEI LF(p29, p30, NC, 1, QEI::X4_ENCODING);
QEI RF(p27, p28, NC, 1, QEI::X4_ENCODING);
QEI LR(p25, p26, NC, 1, QEI::X4_ENCODING);
QEI RR(p23, p24, NC, 1, QEI::X4_ENCODING);
// Ticker to repeatedly call the speed calculation function
Ticker SpeedTicker;
// Constants
float max_rpm = 150.0;
int cpr = 4288;
float threshold = 0.01; //threshold below which speed differences won't be corrected
float K = 0.5;  //control variable for automatic speed control
// state variables
float LF_percent = 0.0;
float RF_percent = 0.0;
float LR_percent = 0.0;
float RR_percent = 0.0;
int lf_pulses = 0;
int rf_pulses = 0;
int lr_pulses = 0;
int rr_pulses = 0;
float correctedSpeeds[4];   //Storage for corrected speed commands
char MODE = 'M';
char CurrentError1 = 'N';
char CurrentError2 = 'N';
// Functions for checking speed:
void CalcSpeed();
float PulsesToPercent(int pulseCount);
void AutoStop();
void Brake();
void CommandLoop();
void Correct();
float AutoCorrect(float intended, float real);
void FloatsToBytes(float lf, float rf, float lr, float rr);
char GetErrorCode(Serial qik);

int main()
{
    pc.baud(115200);
    SpeedTicker.attach(&CalcSpeed, 0.01);  //Call the speed calculation code 100 times a second
[should be fast enough?]
    execute.attach(&CommandLoop, 0.1);
```

```
    qik1.baud(38400);
    qik2.baud(38400);
    while(1)
    {
        char command = pc.getc();
        if (command == 'G')
        {
            pc.printf("%f|%f|%f|%f|%c|%c\n", LF_percent, RF_percent, LR_percent, RR_percent,
CurrentError1, CurrentError2);
        }
        else if (command == 'C')
        {
            //Get command values
            float temp_lf = 0.0;
            float temp_rf = 0.0;
            float temp_lr = 0.0;
            float temp_rr = 0.0;
            pc.scanf("%f|%f|%f|%f\n", &temp_lf, &temp_rf, &temp_lr, &temp_rr);
            LF_speed = temp_lf;
            RF_speed = temp_rf;
            LR_speed = temp_lr;
            RR_speed = temp_rr;
            led4 = 1;
            autostop.attach(&AutoStop, 1.0);
        }
        else if (command == 'B')
        {
            Brake();
        }
        else if (command == 'S')
        {
            AutoStop();
        }
        else if (command == 'R')
        {
            pc.printf("Ready!\n");
            AutoStop();
        }
        else if (command == 'A')
        {
            MODE = 'A';
        }
        else if (command == 'M')
        {
            MODE = 'M';
        }
        if (command == 'I')
        {
            pc.printf("M3\n");
        }
    }
}

void CalcSpeed()
{
    lf_pulses = LF.getPulses();
    LF.reset();
    rf_pulses = RF.getPulses();
    RF.reset();
    lr_pulses = LR.getPulses();
    LR.reset();
    rr_pulses = RR.getPulses();
    RR.reset();
    float temp_lf = PulsesToPercent(lf_pulses);
    float temp_rf = PulsesToPercent(rf_pulses);
    float temp_lr = PulsesToPercent(lr_pulses);
    float temp_rr = PulsesToPercent(rr_pulses);
    LF_percent = temp_lf;
    RF_percent = temp_rf;
```

```
    LR_percent = temp_lr;
    RR_percent = temp_rr;
}

float PulsesToPercent(int pulseCount)
{
    float pulses_per_min = (((float) pulseCount) / .01) * 60.0;
    float rpm = pulses_per_min / cpr;
    float percent = (rpm / max_rpm);
    return percent;
}

void AutoStop()
{
    //pc.printf("Autostopping NOW!\n");
    correctedSpeeds[0] = 0.0;
    correctedSpeeds[1] = 0.0;
    correctedSpeeds[2] = 0.0;
    correctedSpeeds[3] = 0.0;
    LF_speed = 0.0;
    RF_speed = 0.0;
    LR_speed = 0.0;
    RR_speed = 0.0;
    led4 = 0;
}

void FloatsToBytes(float lf, float rf, float lr, float rr)
{
    if (lf >= 0.0)
    {
        commands[0] = 0x88;
    }
    else
    {
        commands[0] = 0x8a;
    }
    commands[1] = (char) (abs(127 * lf));
    if (rf >= 0.0)
    {
        commands[2] = 0x8c;
    }
    else
    {
        commands[2] = 0x8e;
    }
    commands[3] = (char) (abs(127 * rf));
    if (lr >= 0.0)
    {
        commands[4] = 0x88;
    }
    else
    {
        commands[4] = 0x8a;
    }
    commands[5] = (char) (abs(127 * lr));
    if (rr >= 0.0)
    {
        commands[6] = 0x8c;
    }
    else
    {
        commands[6] = 0x8e;
    }
    commands[7] = (char) (abs(127 * rr));
}

void Brake()
{
    qik1.putc(0x86);
```

```
    qik1.putc(0x7f);
    qik1.putc(0x87);
    qik1.putc(0x7f);
    qik2.putc(0x86);
    qik2.putc(0x7f);
    qik2.putc(0x87);
    qik2.putc(0x7f);
}

void Correct()
{
    // This is where actual speed control of the wheels is managed!
    led2 = 0;
    led3 = 0;
    if (MODE == 'M')
    {
        //manual mode with no automatic corrections
        correctedSpeeds[0] = LF_speed;
        correctedSpeeds[1] = RF_speed;
        correctedSpeeds[2] = LR_speed;
        correctedSpeeds[3] = RR_speed;
    }
    else if (MODE == 'A')
    {
        //automatic mode with corrections
        correctedSpeeds[0] = AutoCorrect(LF_speed, LF_percent);
        correctedSpeeds[1] = AutoCorrect(RF_speed, RF_percent);
        correctedSpeeds[2] = AutoCorrect(LR_speed, LR_percent);
        correctedSpeeds[3] = AutoCorrect(RR_speed, RR_percent);
    }
}

float AutoCorrect(float intended, float real)
{
    //Compute a corrected command value based on the intended speed of a wheel and the real speed
of the wheel
    if (intended == 0.0)
    {
        return 0.0;
    }
    else if (abs(intended - real) < threshold)
    {
        //if we're close enough, we don't want to continuously correct and stress the motors/motor
controllers/power system
        return intended;
    }
    else if (intended > real)
    {
        //otherwise, we need to correct up
        float error = abs(intended - real);
        float corrected = intended + (error * K);
        if (corrected > 1.0)
        {
            //if we're trying to drive out of bounds, flash the LEDs and drive to maximum
            corrected = 1.0;
            led2 = 1;
            led3 = 1;
        }
        return corrected;
    }
    else if (intended < real)
    {
        //otherwise, we need to correct down
        float error = abs(intended - real);
        float corrected = intended + (error * K);
        if (corrected < -1.0)
        {
            //if we're trying to drive out of bounds, flash the LEDs and drive to maximum
            corrected = -1.0;
```

```
            led2 = 1;
            led3 = 1;
        }
        return corrected;
    }
    return intended;
}

char GetErrorCode(Serial qik)
{
    //Gets the error code from the provided motor controller
    qik.putc(0x82);
    char error = qik.getc();
    char error_code = 'N';
    if (error & 0x01 > 0)
    {
        error_code = 'M';
    }
    if (error & 0x02 > 0)
    {
        error_code = 'M';
    }
    if (error & 0x04 > 0)
    {
        error_code = 'O';
    }
    if (error & 0x08 > 0)
    {
        error_code = 'O';
    }
    if (error & 0x10 > 0)
    {
        error_code = 'S';
    }
    if (error & 0x20 > 0)
    {
        error_code = 'C';
    }
    if (error & 0x40 > 0)
    {
        error_code = 'F';
    }
    if (error & 0x80 > 0)
    {
        error_code = 'T';
    }
    return error_code;
}

void CommandLoop()
{
    led1 = !led1;
    Correct();
    FloatsToBytes(correctedSpeeds[0], correctedSpeeds[1], correctedSpeeds[2], correctedSpeeds[3]);
    //CurrentError1 = GetErrorCode(qik1);
    //CurrentError2 = GetErrorCode(qik2);
    qik1.putc(commands[0]);
    qik1.putc(commands[1]);
    qik1.putc(commands[2]);
    qik1.putc(commands[3]);
    qik2.putc(commands[4]);
    qik2.putc(commands[5]);
    qik2.putc(commands[6]);
    qik2.putc(commands[7]);
}
```

Sensor Control Microcontroller (ARM Cortex-M0)

```
#include "mbed.h"

//LEDs for visual debugging
DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
DigitalOut led4(LED4);
//Serial Port for connection to host PC
Serial pc(USBTX, USBRX);
//analog 3-axis Accelerometer Pins
DigitalOut accel_self_test(p21);
AnalogIn accel_X(p18);
AnalogIn accel_Y(p19);
AnalogIn accel_Z(p20);
//analog 1-axis Gyroscope pins
DigitalOut gyro_high_pass_filter_reset(p23);
DigitalOut gyro_power_down(p24);
DigitalOut gyro_self_test(p22);
AnalogIn gyro_Z(p17);
//Bumper sensor pins
DigitalIn LFF(p5);
DigitalIn LFS(p6);
DigitalIn LRS(p7);
DigitalIn LRB(p8);
DigitalIn RFF(p9);
DigitalIn RFS(p10);
DigitalIn RRS(p11);
DigitalIn RRB(p12);
//Value storage
char bumpers[8];
//Functions
float sample_X_accel();
float sample_Y_accel();
float sample_Z_accel();
float sample_Z_gyro();
void GetBumpers();
void Publish();

int main()
{
    LFF.mode(PullDown);
    LFS.mode(PullDown);
    LRS.mode(PullDown);
    LRB.mode(PullDown);
    RFF.mode(PullDown);
    RFS.mode(PullDown);
    RRS.mode(PullDown);
    RRB.mode(PullDown);
    pc.baud(115200);
    while(1)
    {
        char command = pc.getc();
        if (command == 'G')
        {
            Publish();
        }
        if (command == 'I')
        {
            pc.printf("M0\n");
        }
    }
}

void Publish()
{
    float temp_X_A = sample_X_accel();
```

```
    float temp_Y_A = sample_Y_accel();
    float temp_Z_A = sample_Z_accel();
    float temp_Z_R = sample_Z_gyro();
    GetBumpers();
    pc.printf("%f|%f|%f|%f$%c|%c|%c|%c|%c|%c|%c|%c\n", temp_X_A, temp_Y_A, temp_Z_A, temp_Z_R,
bumpers[0], bumpers[1], bumpers[2], bumpers[3], bumpers[4], bumpers[5], bumpers[6], bumpers[7]);
}

void GetBumpers()
{
    bumpers[0] = LFF.read();
    bumpers[1] = LFS.read();
    bumpers[2] = LRS.read();
    bumpers[3] = LRB.read();
    bumpers[4] = RFF.read();
    bumpers[5] = RFS.read();
    bumpers[6] = RRS.read();
    bumpers[7] = RRB.read();
}

float sample_X_accel()
{
    return accel_X.read();
}

float sample_Y_accel()
{
    return accel_Y.read();
}

float sample_Z_accel()
{
    return accel_Z.read();
}

float sample_Z_gyro()
{
    return gyro_Z.read();
}
```

Appendix C – Final Design Budget

Item	Source	Cost
Onboard Computer (High-level control & data processing)	Model currently unknown, dependent on CS funding	$700
Kinect Sensor (Primary imaging & depth sensor)	Newegg Item # N82E16874103199	$150
4x Mecanum Wheels (allow for movement in all directions)	AndyMark Part # AM-0137	$260
Mbed microcontroller (Cortex-M3) (control microcontroller)	Sparkfun SKU # DEV-09564	$60
Mbed microcontroller (Cortex-M0) (sensor microcontroller)	Sparkfun SKU # DEV-11045	$45
1 Axis Gyroscope (Inertial measurement & wheel-slip detection)	Sparkfun SKU # SEN-10100	$20
3 Axis Accelerometer (Inertial measurement & wheel-slip detection)	Sparkfun SKU # SEN-09269	$25
Bumper Sensors (4)	Vex Robotics P/N: 276-2159	Total $52: $13 (4)
2x Pololu Dual 10A 12V Motor Controllers	Pololu Part # 1112	$150
4x Pololu 12V Motors with encoders	Pololu Part # 1446	$160
80/20 Extruded Aluminum 1” x 1” x 72” Lightweight (5)	Drillspot Model # 1010	$250
80/20 Mounting Hardware Corner Bracket (28) Fasteners 15-pack (8)	Drillspot Model # 40CB4801 Model # 3320-15	Total $160: $50 $110
Aluminum Sheet 1/8”Thick 24” x 48” Plate (2)	Drillspot Model # 3DRZ9	$271
2x 18Ah 12V Batteries Sealed Lead-Acid Battery	Interstate Batteries Item # BSL1116	$190
Battery Charger & Power Converter Intelligent DC-DC Converter (2) PicoUPS-120-ATV Charger (2)	Mini-Box Item # DCDC-USB Item # picoUPS-120	Total $190: $60 ($120) $35 ($70)
Power Cables & Power Supply P4-P4 12V Power Cable (6)	Mini-Box Item # P4-12V	Total $12: $12
15V Power Supply (320W)	TRC Electronics Part # SP-320-15	$61.32
Miscellaneous Connectors & Cables	Various	$60
Helical Beam Shaft Couplings (4)	McMaster-Carr Item # 9861T529	$120
Double Shielded Ball Bearing (8)	McMaster-Carr Item # 57155K376	$40
Aluminum (Alloy 6061) 3/16" by 2" by 3'	McMaster-Carr Item # 8975K533	$13

Total		$2990

Resources

1. Cousins, S.; Gerkey, B.; Conley, K.; Garage, W.; , "Sharing Software with ROS [ROS Topics]," Robotics & Automation Magazine, IEEE , vol.17, no.2, pp.12-14, June 2010
   doi: 10.1109/MRA.2010.936956
   URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5480439&isnumber=5480272

2. Kyung-Lyong Han; Oh-Kyu Choi; Jinwook Kim; Hyosin Kim; Lee, J.S.; , "Design and control of mobile robot with Mecanum wheel," ICCAS-SICE, 2009 , vol., no., pp.2932-2937, 18-21 Aug. 2009
   URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5333831&isnumber=5332438

3. Jianqiang Jia; Weidong Chen; Yugeng Xi; , "Design and implementation of an open autonomous mobile robot system," Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on , vol.2, no., pp. 1726- 1731 Vol.2, April 26-May 1, 2004
   doi: 10.1109/ROBOT.2004.1308073
   URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1308073&isnumber=29025

4. El-Medany, W.M.; Abuaesh, A.M.; Abuaesh, N.M.; , "An efficient car like-mobile robot design," EUROCON 2009, EUROCON '09. IEEE , vol., no., pp.996-1001, 18-23 May 2009
   doi: 10.1109/EURCON.2009.5167756
   URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5167756&isnumber=5167592

5. do Nascimento, T.P.; da Costa, A.L.; Paim, C.C.; , "AxeBot Robot the Mechanical Design for an Autonomous Omnidirectional Mobile Robot," Electronics, Robotics and Automotive Mechanics Conference, 2009. CERMA '09. , vol., no., pp.187-192, 22-25 Sept. 2009
   doi: 10.1109/CERMA.2009.77
   URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5341992&isnumber=5341933

6. Richard Weiss and Isaac Overcast. 2008. Finding your bot-mate: criteria for evaluating robot kits for use in undergraduate computer science education. J. Comput. Small Coll. 24, 2 (December 2008), 43-49.

7. Jennifer S. Kay. 2010. Robots in the classroom ... and the dorm room. J. Comput. Small Coll. 25, 3 (January 2010), 128-133.

8. Anne-Marie Eubanks, Robert G. Strader, and Deborah L. Dunn. 2011. A comparison of compact robotics platforms for model teaching. J. Comput. Small Coll. 26, 4 (April 2011), 35-40.

9. Arnaud Ramey, Victor Gonzalez-Pacheco, and Miguel A. Salichs. 2011. Integration of a low-cost RGB-D sensor in a social robot for gesture recognition. In Proceedings of the 6th international conference on Human-robot interaction (HRI '11). ACM, New York, NY, USA, 229-230. DOI=10.1145/1957656.1957745 http://doi.acm.org/10.1145/1957656.1957745

10. Dave Touretzky - chiara-robot.org, www.chiara-robot.org, 2011.