```
################################################################################
#                                                                              #
#                          ADVANCED EDUCATIONAL ROBOT                          #
#                                                                              #
#   Setup instructions, FAQ, and usage instructions for the ROS package        #
#                                                                              #
#   AER software is licensed under a 2-clause BSD license (the same as ROS)    #
#   and the AER documentation is licensed under a Creative Commons CC-BY-SA    #
#   license (Attribution + Share Alike).                                       #
#                                                                              #
################################################################################

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

########################################
#                                      #
#   Section 1:  HARDWARE SETUP         #
#                                      #
########################################

-----
1.A     Platform Microcontroller Setup
-----

    Setting up the platform microcontroller is fairly straigtforward, and
    consists of four main steps:

    I)  Program the microcontroller with the provided binary image.
        "AER_Platfrom_Micro.bin" is the file name of the provided binary.
        Using the Mbed microcontroller, this is accomplished by copying the
        binary to the Mbed, which will appear as a USB storage device when
        connected to a computer. No programming cables/devices/JTAG adapters
        are needed to program the device. When programming is complete, press
        the reset button on the microcontroller to load the new binary.

    II) Connect the serial ports of the motor controllers to the
        microcontroller. By default, both Qik motor controllers are connected
        to the microcontroller using a special Y cable that connects RX, TX
        GND and +5V out from the Qiks to the microcontroller. This cable is
        specially keyed and will only connect one way.

    III)Connect the motor encoders to the microcontroller. This is done using
        the provided 10-pin keyed cable that connects +5V, GND, and the A and
        B pins of each encoder. If this is connected incorrectly, you will
        likely receive a negative value from an encoder while moving forwards.

    IV) Connect the microcontroller to the robot's control computer using USB
        and note the serial port assigned to it. To do so, run the following in
        terminal and note the ID strings returned:

        $ ls /dev/serial/by-id

        One of these IDs is the unique ID assigned to the Mbed-M3, another is
        assigned to the Mbed-M0. Figure out which one is which (hint - both
        microcontrollers will reply with their name if given a 'I' character at
        115200 baud). Use these IDs to set the parameters in the launch scripts.

-----
1.B     Sensor Microcontroller Setup
-----

    Setting up the sensor microcontroller is fairly straigtforward, and
    consists of three main steps:

    I)  Program the microcontroller with the provided binary image.
        "AER_Sensor_Micro.bin" is the file name of the provided binary.
        Using the Mbed microcontroller, this is accomplished by copying the
        binary to the Mbed, which will appear as a USB storage device when
        connected to a computer. No programming cables/devices/JTAG adapters
        are needed to program the device. When programming is complete, press
        the reset button on the microcontroller to load the new binary.

    II) Connect the bumper sensors to the microcontroller. This is done using
        the provided 9-pin cable that connects +3.3V and 8 pins (one from each
        of the sensors).

-----
1.C     Power Supply Setup
-----

    The AER platform uses two DCDC-USB power supplies made by Mini-Box to convert
    the varying input voltages from the robot's power supply to the stable 12V DC
    used by the motors and accessories and to the 19V DC used by the onboard PC.
    A USB port is provided on each power supply to monitor the input and output
    voltages; we care about the input voltage, since we can determine whether the
    vehicle is charging (15V input) or running on batteries (~12V input), and we
    can tell how charged the batteries are.

    Using the DCDC-USB is complicated bcause the manufacturer's drivers fail to
    build in current Linux versions, and the only available 3rd-party drivers
    require root permissions to run successfully. This is because Linux restricts
    userspace access to non-standard USB devices, and the DCDC-USB is a non-
    standard USB HID device. Obviously, we can't, and really don't, want to run a
    node as root because we don't want to expose root permissions to ROS.

    1) First, we need to setup the 3rd-party driver. IF AND ONLY IF you are using
    a recent 64-bit (x86_64) Linux system, you can skip this step. If not, delete
    the "debugtool" executable in the ROS package's directory and cd to the
    "DCDC_Components" directory. If you do not have the program "scons" installed,
    install it now with the following command:

    $ sudo apt-get install scons

    Then, run the following command to build the driver:

    $ scons

    When that completes, copy the "debugtool" executable to the parent directory.

    2) We solve this problem by changing the device permissions in Linux to allow
    userspace access to the power supply, which requires one changes to the
    host's configuration:

    As ROOT, we make a new file "/etc/udev/rules.d/dcdc-usb.rules" with the
    following text between the horizontal lines:

-------------------------------------------------------------
# make the dcdc-usb device writeable
ATTRS{idProduct}=="d003",ATTRS{idVendor}=="04d8",MODE="666",GROUP="input"
-------------------------------------------------------------

    This configures the UDEV system that handles device connections to allow
    user-space access to the device.

    We're not quite done yet, though. The existing 3rd-party driver compiles to a
    single executable that prints to the terminal--user friendly, but not
    particularly usefull from our persepective, since we need to store the output
    and publish it in ROS. That's why we have a python class "DCDC_USB" in the
    file "DCDC_USB_Monitor.py" that handles running the 3rd party code and
    retrieving the voltage data, which it returns in a dictionary with entries:

    "input" = input voltage to the DC-DC converter
    "output" = regulated output voltage from the DC-DC converter
    "ignition" = the measured ignitition voltage [we don't use this]
```

This class is used in the "AER_Platform_Driver" node to read and publish the state of the vehicle's power system, so that other nodes can monitor the status of batteries, charging, and the like.

```
-----
1.D     Qik Motor Controller Setup
-----
```

Setting up the motor controllers is extremely simple: one step that needs to be done to both controllers.

I) Use one of the provided jumper blocks to connect the two pins of the "BAUD2" jumper together. This sets the baud rate of the controllers to a fixed rate of 38,400 baud, which allows the controllers to skip their automatic baud detection cycle.

```
-----
1.E     Kinect RGB+D Sensor Setup
-----
```

The OPENNI drivers for ROS should already be installed on the host computer; if they aren't, you can install them with:

```
$ sudo apt-get install ros-<ros version>-openni-kinect
```

replacing <ros version> with the appropriate version (such as electric)

As the Kinect sensor requires +12V DC to augment the +5V over USB, connect the red JST-RCY connector on the Kinect's adaptor cable to the matching red JST-RCY connector on the 12V harness that connects to the motor controllers.

To ensure that the Kinect sensor works properly, it needs as much USB bandwidth as possible. As the onboard computer has 4 USB 3.0 ports, the easiest way to provide sufficient bandwidth is to connect the Kinect's USB connector to one of the blue USB 3.0 ports.

```
-----
1.F     Joystick Setup
----
```

The ROS joystick driver must be installed to run all teleop nodes; if it is not already installed, do so with:

```
$ sudo apt-get install ros-<ros version>-joystick-drivers
```

This driver will work for all _standard_ USB joysticks; we have tested it with wired and wireless Microsoft XBOX 360 controllers, and the default axis and button mappings in the teleop nodes are set up specifically for these controllers. If you are using a different controller, test first to make sure the mappings are correct.

*** NOTE THAT THE DEFAULT ROS DRIVER DOES NOT SUPPORT PS3 CONTROLLERS ***

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

########################################
#                                      #
#   Section 2:  SOFTWARE SETUP         #
#                                      #
########################################

-----
2.A     ROS Environment Setup
-----
```

Make sure ROS can find the AER package by changing the ROS_PACKAGE_PATH.

```
    To do so permanently, do the following all on one line:

    $ echo "export ROS_PACKAGE_PATH=/path/to/package:$ROS_PACKAGE_PATH"
     >> ~/.bashrc

    followed by:

    $ ~/.bashrc

    This changes you ROS_PACKAGE_PATH so that rospack, roscd, and ROS tab
    completion will work for the AER package.

-----
2.B     Build the AER Package
-----

    To be correct, as the entire package consists of python code, it requires no
    "building" in the same way it would if it consisted of C++ code. However, to
    satisfy ROS and set up the services and topics, we need to use ROSMAKE to set
    everything up properly. Run the following commands:

    $ make clean [almost always, you can ignore any errors here]
    $ rosmake

    The ROSMAKE command should complete successfully; if it does not, correct any
    errors and try again until the package builds successfully.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#########################################
#                                       #
#   Section 3:  RUNNING THE SOFTWARE    #
#                                       #
#########################################

-----
3.A     Testing the Teleoperation System
-----

    I)  Connect a USB jostick to the host computer

    II) Power on the robot, making sure the E-STOP switch is up

    III)Run the following in the terminal:

    $ ./Teleop.sh

    IV) Wait for all nodes to finish starting up, and then run the following
    in a new terminal window:

    $ rosnode list

    You should see "/AER_Driver", "/AER_teleop" and "/joy_node" in this list. If
    not, something has gone wrong.

    V)  To drive the robot, press and hold the green "A" button on the jostick
    (this serves as a deadman's switch) and use the left analog stick and the
    triggers to drive the robot. The left analog stick provides linear X and Y
    driving commands, and the triggers provide angular Z steering commands. If
    you let go of the green button, the robot will stop immediately.

    VI) If, for any reason, the robot begins to operate in an uncommanded fashion
    and/or fails to stop and/or threatens other people and/or equipment, trigger
    the E-STOP killswitch by firmly striking the red button on the upper rear of
    the robot. In addition to killing all power to the drive motors, this also
    kills all power to the 12V DC accessories bus (this includes the Kinect, so
    any running nodes using the kinect will probably crash at this point).
```

```
-----
3.B     Running the System Driver
-----

    I)  Power on the robot, making sure the E-STOP switch is up

    II) Run the following in the terminal:

    $ ./DriverCore.sh

    III)Wait for all nodes to finish starting up, and then run the following
    in a new terminal window:

    $ rosnode list

    You should see "/AER_Driver" in the list of nodes. If not, something has gone
    wrong.

-----
3.C     Running the Rescue Driver
-----

    There are two ways to run the rescue driver:

    1)

    Plug in the provided wired USB game controller. (That's it!)

    2)

    I)  Run the following in the terminal:

    $ ./Rescue.sh

    II) Wait for all nodes to finish starting up, and then run the following
    in a new terminal window:

    $ rosnode list

    You should see "/AER_RescueDriver", "/AER_teleop" and "/joy_node" in this
    list. If not, something has gone wrong.

    III)To drive the robot, press and hold the green "A" button on the jostick
    (this serves as a deadman's switch) and use the left analog stick and the
    triggers to drive the robot. The left analog stick provides linear X and Y
    driving commands, and the triggers provide angular Z steering commands. If
    you let go of the green button, the robot will stop immediately.

    IV) If, for any reason, the robot begins to operate in an uncommanded fashion
    and/or fails to stop and/or threatens other people and/or equipment, trigger
    the E-STOP killswitch by firmly striking the red button on the upper rear of
    the robot. In addition to killing all power to the drive motors, this also
    kills all power to the 12V DC accessories bus (this includes the Kinect, so
    any running nodes using the kinect will probably crash at this point).

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

########################################
#                                      #
#   Section 4:  DRIVER COMPONENTS      #
#                                      #
########################################

-----
4.A     PlatformComponents.py
-----
```

```
    //Not an ROS node

    Provides a variety of Python classes that handle hardware communication,
    implement the drive algorithms for the mecanum wheels, and provides a class
    for storing the state of sensor values.

-----
4.B     TestComponents.py
-----

    //Not an ROS node

    Provides an emulator class for the real hardware control class in
    PlatformComponents.py that allows for testing and debugging without real the
    real robot hardware.

-----
4.C     teleop.py
-----

    <Publishes> geometry_msgs/Twist (/cmd_vel)
    <Subscribes> sensor_msgs/Joy (/joy)

    Converts joystick commands into 'command-velocity' commands that drive the
    robot.

-----
4.D     AER_Driver.py
-----

    <Publishes> std_msgs/String (/AER_status)
    <Subscribes> geometry_msgs/Twist (/cmd_vel)

    Actually drives the robot from 'command-velocity' commands, and reports the
    status of various robot subsystems.

-----
4.F     DCDC_USB_Monitor.py
-----

    //Not an ROS node

    Python class that reads data from the DCDC-USB power converter and returns it
    as a dictionary in the format of:

    {'input':<float>, 'ignition':<float, 'output':<float>}

    Each float is a positive DC voltage value. By monitoring the input voltage,
    you can check the charge state of the batteries and check if the vehicle is
    properly charging. - See note above in Section 1.C

-----
4.G     debugtool
-----

    //Not an ROS node

    Third-party binary that handles access to the DCDC-USB power converter. This
    is called by the DCDC_USB_Monitor.py to handle hardware-level USB data
    transfer and parsing.

-----
4.H     Kinect_Depth_Navigator.py
-----

    <Publishes> geometry_msgs/Twist (/cmd_vel)
    <Subscribes> sensor_msgs/Image (/camera/depth/image_rect)

    Reads in data from the Kinect sensor, processes it, and attempts to drive in
    the direction of the fewest obstacles (i.e., the longest straight-ahead
    distance). If this node fails, it's probably because the OPENNI drivers
    'helpfully' return nan (not-a-number) for out-of-bounds data.

-----
4.I     Teleop.sh
-----

    //Not an ROS node

    Shell script that kills all running nodes, restarts roscore, and launches all
    nodes needed for full joystick teleoperation.

-----
4.J     KinectNavigate.sh
-----

    //Not an ROS node

    Shell script that kills all running nodes, restarts roscore, and launches all
    nodes necessary for navigation with Kinect_Depth_Navigator.py

-----
4.K     Rescue.sh
-----

    //Not an ROS node

    Shell script that kills all running nodes, restarts roscore, and launches the
    minimum number of nodes to handle backup joystick teleoperation of the robot.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#########################################
#                                       #
#   Section 5:  USAGE INFORMATION       #
#                                       #
#########################################

-----
5.A     Design Parameters
-----

    The Advanced Educational Robot platform was designed to fullfil the following
    criteria:

    - Flexibility

    - Ease of use

    - Manouverability

    - Cost-efficiency

    In particular, it was designed to offer better performance, flexibility,
    manouverability, and be easier to use than common educational robots such as
    Mobile Robot's Pioneer series robots.

    In light of this, we designed the robot with some basic design parameters:

    - Size: small enough to fit through doorways, yet large enough to provide a
            stable base for sensors, arms, and other accessories mounted atop the
            vehicle.
    - Payload: an unloaded (no external sensors or additional equipment) weight
               of approximately 25-30 kg (~60 pounds) including batteries, and a
               maximum loaded weight of 40-45 kg (~100 pounds).

    - Performance: maximum forwards velocity of ~1.5 m/s, which is about the same
                   speed as a fast human walk. Ideally, maximum sideways velocity
                   would be the same, but different types of flooring impair this
                   kind of motion, so sideways velocity was only designed to be
                   about 0.5 m/s to 1 m/s depending on floor surface.

    - Computers: a Core i3-based onboard computer is provided for high-level
                 control of the vehicle. Direct control and communication with
                 vehicle sensors and actuators is accomplished through two Mbed
                 microcontrollers, one of which (the blue one) is based on the
                 ARM Cortex-M3 microcontroller, and the other (the yellow one) is
                 based on the ARM Cortex-M0 microcontroller.

    - Sensors: hall-effect quadrature encoders are provided on all four wheels to
               measure wheel speeds, a 3-axis analog accelerometer (an Analog
               Devices ADXL335) and a 1-axis analog gyroscope (an STMicroelectronics
               LY530AL) for measuring vehicle motion, voltage sensors built into
               the DC-DC power converters to monitor the onboard power use and
               batteries, and a Kicrosoft Kinect RGB+D sensor for color and depth
               imaging.

    - Software: all software that runs on the microcontrollers is written in C++
                with extremely few external libraries (to make it easier to
                port to different microcontroller platforms), while almost all of
                the higher-level software that makes us the ROS driver stack is
                written in Python. The sole exception to this is the 3rd-party
                driver to the DCDC-USB converters, which is written in C.

                Currently, the Advanced Educational Robot is designed to be fully
                supported in the Robotic Operating System (ROS), but we intend to
                provide support for Microsoft's Robotics Developer Studio (MRDS)
                at some point in the future so that users can choose between the
                two environments depending on their needs/preferences.

-----
5.B     Onboard Power Supply & Distribution
-----

    The robot is designed to supply two power busses: a 19V DC bus for the
    onboard computer and a 12V DC bus for the motors, sensors, and accessories.
    Both busses are supplied by DC-DC converters that regulate the varying
    input voltages from the charger and batteries. These converters can currently
    supply up to 190 Watts to the computer and 120 Watts to the 12 motor bus. If
    more power over the 12V bus is needed, another DC-DC converter can be added,
    but keep in mind that any increases in power consumption lead directly to
    reductions in battery life.

-----
5.C     Building & Adding Accesories
-----

    As the AER platform was designed explicitly for flexibility of use, it makes
    sense to provide an example in the documentation of how to design, implement,
    and integrate an accessory on the robot. In particular, we describe how to
    add an arm to the robot so that the robot can pick up items off the floor.

    First, we design the arm itself: we know it must be long enough and flexible
    enough to reach objects on the floor around the robot, and we know it must be
    accurate and easy enough to control so that it will be possible to accurately
    grab items. This leads us to choose a design based on servos for the joint
    actuators, a camera mounted near the gripper to distinguish objects, and
    some pressure/temperature/vibration sensors to be mounted on the gripper
    itself to allow investigation of objects.
    Working from these design constraints, we design the arm with a USB servo
    controller, a USB-connected microcontroller to handle sensor interfacing, and
    a USB camera to handle imaging. Similarly, we choose servos that take 12V DC
    power supply, or we add a DC-DC converter that can provide the correct input
    voltage for the servos. Note that we have limited ourselves to two standard
    interfaces: USB for data, 12V DC for power.

    Communication over busses like CAN, SPI, I2C, RS485, or TTL Asyncronous
    Serial are best handled through USB adaptors or the use of USB-connected
    microcontrollers that implement these interfaces. An ethernet interface to
    the onboard computer is available, we recommend, however, not using it for
    anything other than special sensors which need the higher data rate provided
    by 1 Gb/s ethernet.

    Construction of the arm and any independent testing should be done separately
    from the robot, since it will always be easier to test and debug in a more
    controlled environment. When the time comes to attach the arm to the robot,
    all that's needed is that the arm have a set of mounting holes that match the
    mounting holes provided on the top plate of the robot. Simply attach the arm
    with bolts to the top plate of the robot and connect the data (USB) and power
    (12V DC) connectors to the provided connections on the top of the robot.

-----
5.D     Using Indoor/Outdoor Wheels
-----

    While the provided mecanum wheels allow impressive manouverability, they are
    extremely sensitive to floor conditions and are almost entirely unsuitable
    for outdoor use, even on paved surfaces. We recommend that if you need to use
    your robot in an indoor/outdoor setting that you swap the mecanum wheels for
    standard pneumatic tires which will provide better traction, impact
    resistance, and more reliable control. Obviously, doing so will prevent you
    from using the full holonomic drive capabilities of the platform, which is
    why we provide a "TankDrive" class in "PlatformComponents.py" specifically to
    handle the case of using normal wheels on the robot. In place of normal
    wheels, one could also substitute tracks, and such drive behavior can also
    be controlled using the "TankDrive" class.


-----
5.E     Adding Suspension
-----

    Because the robot was designed originally for indoor use on fairly level
    surfaces, it was not designed with a suspension system. We have noticed in
    trials on the robot that this design choice results in more wheel slippage
    than expected, and that adding a suspension system would probably improve
    performance.

    The best way to add a suspension system to the robot is to completely remove
    the current drive system and cut away all of the vertical 80/20 members that
    extend below the bottom plate. From this, design new modular drive units that
    contain a wheel, motor, and encoder. With four of these units, they can be
    independently mounted on shock absorbing/suspension mounts, which provides
    the robot with a fully-independent suspension.

    Be warned that redesigning the drive system is by no means a simple process,
    and will probably take months to design, test, revise, and reimplement.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Documentation License: CC+BY+SA "Some rights reserved"

Software License: BSD 2-clause (Copyright AER Developers, All rights reserved)

Authors: Calder Phillips-Grafflin (Union College)
```

```
        Erik Skorina (Union College)
```