ARIA: Getting Started Quickly

Kapil Vyas, Vangelis Metsis, Fillia Makedon
416 Yates Street, 250 Nedderman Hall
Arlington, TX 76019, USA
kapil.vyas@mavs.uta.edu, vmetsis@cse.uta.edu, makedon@uta.edu

ABSTRACT

ARIA is a popular, reliable and powerful C++ Robotics library commonly used at various research labs around the world. In spite of its powerful features, it often presents itself as an intimidating environment for beginners. This paper is an attempt to bridge this gap and help a wide array of researchers who struggle to get started with one of the world's most reliable robotics API. As such, it is an effort towards compiling a general guideline that will allow roboticists, researchers, and hobbyists to quickly get moving with the robot. The goal is to minimize upfront issues, both small and big, and allow users of the library to promptly get to the core task at hand.

Keywords

ARIA, Robots, Mobile Robots, Robotics API

1. INTRODUCTION

1.1 Motivation

The field of Robotics is gaining momentum and with that a wider audience is involved with the growth. Recent advances in other areas of computing such as Artificial Intelligence and Computer Vision has steered it forward and brought it into mainstream media. This era in time can be collated with another one in history, one between computers and operating systems. When computers began to be more accessible and affordable to a bigger population, we saw tremendous evolution of the Operating System. Today, robots just like computers are following the same trend and they too are facing a desperate need to have a reliable architecture to support the multitude of complex programming tasks. One such good library with a well defined architecture and proven track history is the ARIA library. ARIA, developed by Mobile Robots Inc., stands for **Advanced Robotics Interface for Applications**, is a C++ library used to control robots (Pioneer, PeopleBot, PowerBot, AmigoBot) made by the same company. ARIA also provides a number of other libraries that deal with other interesting robotic features such as speech synthesis and recognition (ArSpeechRec_Sphinx), network communication (ArNetworking), and simultaneous localization and navigation (ARNL) libraries. It is an Open Source library that runs on multiple platforms while supporting a number of robots. The download section of the main site is continuously updated with new third party softwares. This alone speaks about the vigorous activity of the users that make use of the library. It comes with a basic simulator and out of the box SLAM support.

1.2 Objectives

This paper shall describe how to set up an environment in Linux and Windows for code compilation, communication with various onboard sensors, how to use other languages besides C++ and will walk over sample programs for common tasks. How to setup G++ compiler(Linux) and Microsoft's Visual Studio (Windows) IDE (integrated development environments) to develop ARIA source code is provided in Section 4. The main intention is to allow users to create an environment on their preferred platform (Linux or Windows) use their favorite programming language (C++, Java or Python) to work on the project they began with at once. Section 6 shows how to use the simulator which can be used to test programs before running them on the actual robot. Section 7 explains all the setup needed to write code in Java and Python using the Java and Python wrappers.

2. RELATED WORK AND CONTRIBUTION

As an increasing number of robots have found their ways into research labs, university courses and in the homes of hobbyists, various issues have arose as a result. One of the principle issue is that developers have to understand the different API for each individual robotics platform and also learn the specific programming language that is used to access the API. Researchers have suggested the need for a common development environment. These [development libraries and software] tools are obviously platform dependent and thus they cannot easily be used for building multi-platform robotic systems [3]. One complaint is that a program cannot be easily ported over to another robotics platform even with slight modifications. One has to first get acquainted with the new platform and redo everything from scratch. There is unfortunately no standardized way to interface with robots [1].

This could be addressed by having an international community create a standard API and companies can agree to support the published specifications. One challenge is that the field of robotics itself covers many areas. Say one robot has a robotic arm and another is loaded with sensors but has no gripper of any kind. These robots could support part of the standard API based on what devices can be availed from them, but then it would again be difficult to support all the methods for an individual device as published in the standard document. This would not only make it cumbersome but create frustration due to partial support. As such, the idea towards a common platform for all robots, if not elusive, is very well placed ahead of our time and would require an elaborate understanding of the complex underlying architecture of different systems. A standard platform for a set of robots that share many features would be more approachable than an all-standard one. Such an attempt for a set of robots is already making headway with the Robot Operating System [12]; which is applicable to a fine set of robots where testing and proper documentation is still lagging behind for some of the robots like Mobile Robots PeopleBot. However, the architecture looks very promising. Again, the catch is that most companies would avoid this idea to begin with; as such it limits their freedom by having to abide by a set standard. Finally, the subset is less than the whole; performance is lost and a smaller set of usability is represented by the common robotics platform across independent robots. Borgström who describes a design and implementation of an adapter layer that integrates the ARIA library and MATLAB admits, "... due to the difference in features between the C++ and MATLAB languages the exposed API is not complete. One conclusion is [that] it is possible to create an at least partial adapter layer [1]."

We approach this problem by narrowing our search for a very reliable and highly robust open-source API that supports a multitude of robot designs and capabilities that can target myriad needs of the robotics field. The API should be accessible using common platforms (Windows and Linux) and should be supported by a set of programming languages that an average programmer is conversant with. The set of robots that it supports must be be priced within a reasonable budget size. We chose ARIA because it closely fits our criteria. It is hard to find a good library with many useful built-in features. The striking feature about ARIA is that the library comes with a simple interface and the developers have provided all kind of source code for working with almost anything imaginable that can be carried out by the robot. However, this itself could be cause enough for beginners to stay away from it and become overwhelmed. The ARIA library is primarily designed for professional developers and is meant to be used as a solid base for large-scale applications [1]. Our premise is that the library is not complex but comprehensive. In the long term we are interested to work with a comprehensive library than one with a small feature set. ARIA lacked a well-written step-by-step guide to get started with the library because it assumed that users of the library were experts in C++. However due to its increasing popularity as reflected by the number of third party softwares available for download; one can see the need for a getting started document which describes all the basic setup and basic sensor access information in one place. Until recently (late 2009) and much to the need for the same cause, Whitbrook wrote on how to use ARIA and Player to program Mobile Robots, which is more than a getting started document and covers various facets of the library in greater detail, which helped in the development of certain sections of this paper[4]. The main contribution is to create a short document that summarizes end-to-end setup of the ARIA library for major platforms (Linux, Windows) using key programming languages (C++, Java, Python) with access to basic functionality (camera, laser, sonar, bumpers and motors) and features (Simulator) in one single place. At the same time, the goal was to avoid mixing code with the explanations so that all the coding artifacts are in accessible place that references all the basic programming steps. The idea behind was to let beginners jump into code if they are interested in seeing things running and delay the explanations for later reference.

3. GETTING STARTED

3.1 Remote Communication Approaches

Mobile Robots provides a number of software downloads ("http://robots.mobilerobots.com/wiki/All_Software") to use with their robots on the Software and downloads section of their website. This paper assumes that all robots come with an onboard computer. One can directly program the on-board computer. However, the on-board computers are small and are fixed to the robot. A remote computer is the best way to work with the robots. There are two ways to approach communication with the on-board compute. The first approach is to use a program such as SSH (Secure Shell) for Linux and Remote Desktop Connection using Windows. Secure Shell (SSH) is a network protocol used to communicate between two networked machines [6]. If this approach is followed, all of these software packages need to be installed on the robot's on-board computer. There is no need to install these packages on the remote machine. On Linux, it is much more sensible to use a preferred editor on the remote machine and transfer the code to the on-board computer and run compilation commands via SSH. On Windows, it is easier to install the preferred editor on the on-board computer. The second approach is to use the remote machine and use a client-server relationship to communicate the two machines. In this case, all software packages must be installed on both machines. A client program needs to be written for the on-board machine which sends requests to a server that is running on the robot's onboard computer. The ArNetworking library is used to write the client and server programs. Beginners who are getting started with ARIA should avoid the second approach. Though later, a beginner may be interested in using a simulator or navigate a robot from one corner of a room to another using a 2D map of the apartment room. Mobile Robots Inc. provides a server program, ARNL guiServer and a client program, Mobile Eyes™, particularly designed for the purpose of simulation and navigation. Details about how to setup a simulation environment and do simultaneous localization and navigation (SLAM) is provided in the "ARNL Installation and Operations Manual" [8]. Beginners can follow directions in that manual if they are planning to do simulation and navigation. Unless otherwise stated explicitly, the rest of this paper assumes the second approach of using a remote computer to communicate via SSH (Linux)/Remote Desktop Connection (Windows) or directly using the onboard computer is followed.

3.2 Main Installation

ARIA is the core installation package needed to program the robots. ARIA version 2.7.2 was used to run all the sample programs in this document. All these installations are done on the robot's on-board computer. The Windows installation is straightforward using the "accept license-click next-done" interface. On Linux the source code can optionally be downloaded in `/usr/src` or on the Desktop. Executing `rpm -i` (Red Hat machine) or `dpkg -i` (Debian machine) will install ARIA at `/usr/local` directory. Linux installs applications that are not part of the official distribution at `/usr/local`. You can skip installation of other packages at this point.

4. ENVIRONMENT SETUP

4.1 Code Compilation on Linux

On Linux, both gcc (GNU Compiler Collection) and g++ compiler version 3.4 or higher are required for ARIA version 2.7.2. Typing gcc -v and g++ -v will let you know if the gcc and g++ compiler are installed or not and will also print the version number. This guide used gcc version 4.4.1 with a C++ compiler. Use `sudo apt-get install gcc` with a to install gcc and `sudo apt-get install g++` to install g++. Debian based Ubuntu comes pre-installed with the gcc compiler but not the g++ compiler. On Debian, the g++ installation will also install any dependent packages. The libstdc++.so.5 is a key library required to compile ARIA's code because ARIA's main libAria.so library depends on it. The recent release of Ubuntu's Karmic Koala has discontinued this library. Thus compiling the code gives error and warning messages, complaining of the missing library. The workaround is to install the missing library from this link. However, when the compilation is run, it will give a warning message about using two conflicting libraries libstdc++.so.5 and libstdc++.so.6. To fix this: explicitly include the file path of the library, that is /usr/lib/libstdc++.so.5 in the compilation command. To compile the ARIA code, you need to use the following command to compile the program in order to create an executable:

```
g++ -g -Wall -I/usr/local/Aria/include
-L/usr/local/Aria/lib -lAria -ldl
-lpthread myFirstPrgrm.cpp -o
myFirstPrgrm
```

The -g and -Wall options will print useful warning and debugging messages. The -lAria, -ldl, and -lpthread are additional referenced libraries that the linker needs to know. In C/C++, whenever you include a header file, the pre-processor replaces the line on which the #include directive appears with the whole content of the specified file. When the compiler sees the code, it is seeing a pre-processed file. The inclusion of a header file does not imply that you are using a compiled implementation of the header file. Including, example.h does not mean that a compiled version of example.cpp is to be linked to create the executable. At the same time there is no such convention that a example.h file will always have an implementation defined in example.cpp. It could be defined in a different extension with a totally different name. In short, you also need to tell the linker to link the appropriate referenced library that the header file is for. The referenced libraries must be compiled. In our case, the referenced libraries have been compiled. Some programs of the ARIA library may give an error: "`undefined reference to 'clock_gettime'`". In this case you may need to add `-lrt` as an additional referenced library. The command would be:

```
g++ -g -Wall -I/usr/local/Aria/include
-L/usr/local/Aria/lib -lAria -ldl
-lpthread -lrt myFirstPrgrm.cpp -o
myFirstPrgrm
```

The –L argument specifies the location for linking the source code with the ARIA library. The –I argument specifies the location of the ARIA header files. The –o option followed by the argument will produce an executable: myFirstPrgrm which can be run by the command: `./myFirstPrgrm`. Instead of typing a lengthy command to compile the program, a onetime makefile can be created. A simple `g++ myFirstPrgrm.cpp` command would compile the program without all the arguments. See Appendix 1.1 on details about creating a makefile.

4.2 Code Compilation on Windows

On Windows, only Microsoft's Visual Studio .NET 2003 and later versions can be used to compile ARIA's source code. See Appendix 1.2 for step-by-step instructions for compiling a program on a Windows platform using Microsoft Visual Studio 2008. Visual Studio is a powerful IDE, and it comes with a very good debugger. Linux users may want to use it as a tool for debugging and later port the code over to Linux. In this case, they should avoid writing a program that depends on Microsoft's Foundation Classes (MFC).

5. ROBOT PROGRAMMATIC ACCESS

5.1 Communicating with the Robot

Every program must begin by having a code segment that sets up the communication between the on-board computer and the robot. The easiest and generic way to communicate with the robot for beginners is to use the ArSimpleConnector class. The other way is to use the ArTcpConnection class which is used to write a more specific connection routine and gives more control over how you connect with the robot [5]. The latter has the benefit that one comes out with an understanding of how the robot communication works but it is way too advanced for beginners and should be avoided when one is just trying to get started. The advantage of using the generic ArSimpleConnector class is that if a simulator is present it first tries to connect to it before attempting to connect to the robot itself. Appendix 1.3 gives the code segment to successfully connect to the robot. Again, as stated earlier, all of this code can be written using an editor on a remote machine and then porting over to the robot, say via SSH using a `scp` command (see below) and then compiling it remotely. It will result in extraneous step if all the setup is repeated on the remote machine as explained earlier.

```
[kapil@remotehost~]$ scp kapil@onboardhost hello.cpp
```

5.2 Communicating with the Robot's Devices

In the code of Appendix 1.3 when the main robot class object (robot) is instantiated, the instance already includes motors but does not include any sensors such as laser, bumpers, sonar and camera. Laser, sonar and bumpers inherit from a single parent class called the ArRangeDevice and the group can be called "Ranged Devices". Therefore they have the same method of connecting to the robot. In appendix 1.4, code shows initialization and connection of the ranged devices and the camera with the main robot object. The ARIA Architecture has been designed in such a way that the ranged devices (sonar, bumpers and laser) can be initialized without requiring the robot object and thus the sensor values can be accessed. The ranged devices need to be explicitly added using the *addRangeDevice()*. In the case of the camera, a non-ranged device, the initialization cannot be made without the robot object. Since the camera is typically connected to the robot microcontroller's auxilliary serial port, and also uses ArRobot task cycle callbacks, a connected and running ArRobot object is required[5]. The section below shows how individual devices can be controlled and their sensor values be accessed.

5.2.1 Motors

The ArRobot object is used to give motor commands to the robot. Taking a look at the ARIA API reveals more than 150 methods that relate to the ArRobot class[5]. It may take a good deal of time to realize which methods are used to fully control the robot's motion. Beginners should not be overwhelmed by this; knowing half a dozen of the available methods should enable maximum control of the robot's motion. Appendix 1.5 summarizes key methods related to motor control.

The API has one good utility class for describing motion (ArAction). It is recommended that the API be read after one becomes familiar to dig for useful utility classes to reduce development time. One such useful utility class is the ArPose class. The ArPose class allows setting up of different positional poses that the robot can move through. The constructor takes three arguments; the (x,y) position and the n^{th} position starting with n=0. As such this class can be used to move the robot through a planned path assuming a certain coordinate frame defined by the programmer. The ArPoseWithTime is inherited from the ArPose class and also ties the time when the pose was taken.

5.2.2 Sonar

Sonar is one of the ranged devices and maps the closest object detected. It assists the robot to stay clear off collisions. Each sonar array has 8 transducers but only one transducer is fired at a time from each of the sonar array[13] from left to right. The acquisition rate for each array is 40milliseconds/transducer (40Hz). A Peoplebot has 8 sonar arrays and to read all the raw sonar values can take about 0.32s. The sensitivity of the sonar is roughly between 0.01-5meters but exact range depends on the ranging rate as set by the position of the sonar-gain potentiometer. The sonar-gain potentiometer's position can alter the range and sensitivity. It is a visible cap just located near the underside of the sonar array. Using a flat screw, turning the cap counter-clockwise makes the sonar less sensitive and is useful for operating in a noisy environment. This also reduces the ability of the robot to see tiny objects because of its reduced sensitivity. But if one is operating in a calm area, high sensitivity gain can be set. However, if the surface is too reflective or there is a heavy carpet underneath, then the sonar will detect the floor as an obstacle. It is advised that the sonar settings be made later and only if needed depending on the severity of the operating environment or a need to get a more refined data set.

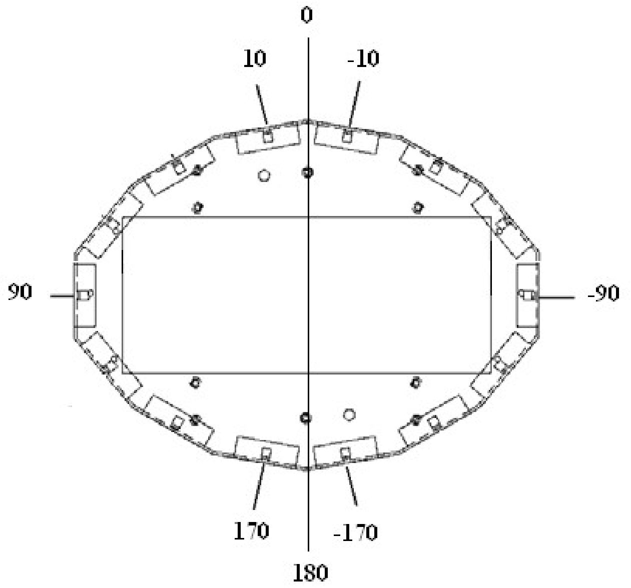

Figure 1: Sonar base showing angular distribution [13]

The ARIA library describes the outward facing area surrounding the sonar as "polar region"[5]. To get a reading from each of the sonar array, one needs to specify exactly the polar region. The polar region is described by a start and end angle in degrees in counterclockwise direction. The front is at zero degrees. Therefore to get sonar values on the front side of the robot, the slice of the polar region would have a start angle at -90 and an end angle of 90. Specifying the other way round (90,-90) would yield values on the backside of the robot. In short (0,10) would mean 10 degrees left of the front side but (10,0) would mean 350 degrees on the remaining area. However, the value returned by the sonar is not a distance measure from the sonar array to the object identified. ARIA's sonar method *getReadingPolar()* adds a constant radius from a fixed center of the robot. The radius can be obtained from the *getRobotRadius()* method of the ArRobot class. Appendix 1.6 has a code snippet that shows accessing the sonar values.

5.2.3 Laser

Laser Rangefinder is another ranged device and also maps the closest object detected. The ArSick class defines the laser object, and just like the ArSonarDevice class for the sonar, they both inherit from the ArRangeDevice class. The way to programmatically access by polar region works the same way as the sonar. See Appendix 1.7.1 for laser-specific code to access values from the range finder. Appendix 1.7.2 describes how to access all the laser values using an iterator which is slightly different code-wise from the sonar; since the former does it more comprehensively, it returns a huge data set than the later.

5.2.4 Bumpers

The bumpers are triggered when an obstacle comes into contact with the robot and is the last line of defense when other obstacle detection sensors have failed. Additionally, Mobile Robot has a safe feature that will stop the robot from moving if any of the bumpers has been physically removed[13].

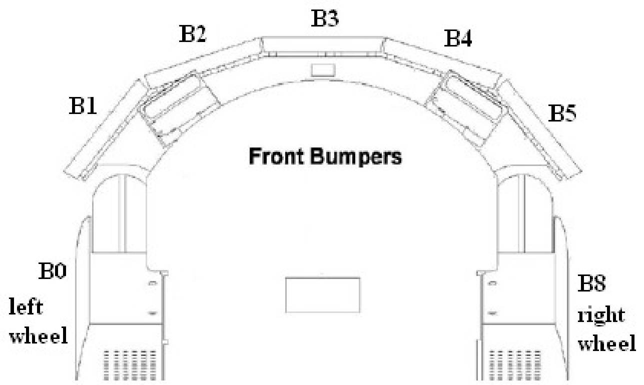

Figure 2: Bumper base showing the bit values of the front bumpers including the left and right wheel; that is bit 3 would represent the center bumper on the front side [13]

Programmatic access of bumpers is simple as calling the *getStallValue()* method of the ArRobot class. This returns an integer whose value in binary format expresses the stall flags of the bumpers, left wheel and right wheel. In a robot with both front and rear bumpers, the binary number will produce 16 significant bits of information and if it has just one set of bumpers, front or rear, then only 9 bits will be of interest. The bits represent the stall flags or indicators for the bumpers, left wheel and right wheel. Assuming a robot with both the front and rear bumpers, 16-bits of information can be retrieved from the integer returned. A simple binary And (&) can be used to access the individual bits. Say, to access the bit zero, we would And it with 2 raised to the bit number.

```
int stallValue = robot.getStallValue();
int bit0 = stallValue & 0;
```

A bit is set to 1 if a bumper or wheel represented by that bit is stalled, otherwise it is set to 0. Bit 0 is the left wheel stall indicator. In the above code, if bit0 has value 1 then the left wheel is stuck and cannot move. Bits 1-7 matches the first bumper, so say for the front bumpers in figure 1, bit 1 would stand for the leftmost bumper,B1 and bit 5 would stand for the rightmost bumper,B5 while bit 6 and 7 does not mean anything. Bit 8 would correspond to the right wheel stall indicator. Bits 9-15 correspond to the second bumper set, say the rear bumpers which also has a set of 5 bumpers, but bit 14 and bit 15 are unused and will have default zero value.

The ArRobot class also provides methods to check if front and rear bumpers are present and also access the number of front and rear bumpers:

```
int numFront, numRear;
if( robot.hasFrontBumpers() )
  numFront =
robot.getNumFrontBumpers();
if( robot.hasRearBumpers() )
  numRear = robot.getNumRearBumpers();
```

5.2.5 Camera

The camera is positioned in such a way that robots with a gripper can use the camera to inspect the contents of gripper. On PeopleBot, the 16X zoom lens color camera is mounted at chest height, such that face detection and object recognition and tracking can be done. The robot's (Canon VC-C4[R] or C50i) cameras come with a programmable robotic pan-tilt base[12]. The ArVCC4 class is used to control the pan, tilt and zoom mechanisms and some other aspects of the camera[5].

The camera operates in two modes; unidirectional and bidirectional. In unidirectional mode, when a command packet is sent by the ArVCC4 class to the camera, it does not return a response; it simply delays for some time to allow the camera to complete processing the command it was sent. In bidirectional mode, when a command packet is sent to the camera, it takes about 300ms for it to generate a response. However, the response does not indicate when the command will be completed, it only gives a binary indication as to if a command will be executed or not. The default constructor sets it to unknown communication (COMM_UNKNOWN) which means it uses a bidirectional mode if a response is received. See Appendix 1.8 for a complete list of key methods to control the pan-tilt base. Before calling the methods, one must check that the camera was successfully initialized:

```
bool check = myCamera.init();
```

It should be noted that the simulator program, MobileSim, does not simulate the programmatic pan-tilt base.

There are many ways to access the images grabbed by the pan-tilt-zoom camera. Video for Linux (V4L) is a core Linux library which underpins many popular Linux applications related to camera such as the GUVCView, Cheese and the GStreamer. Appendix 1.9.1 provides a generic class for Linux developers and employs the Video for Linux (V4L) library. To use the generic class, UVC drivers must be installed [11]. You can use a program like gstreamer and type in `gstreamer-properties` to check if the UVC drivers are installed or not. Appendix 1.9.1.1 provides the header file, Appendix 1.9.1.2 gives the source file and Appendix 1.9.1.3 shows a sample test program that uses the generic camera class. The class is written in such a way that it can be used to grab images from multiple cameras. As such, it is making use of Linux threads. Appendix 1.9.2 provides a generic class for accessing camera in a Windows environment.

Other third party resources are ACTS and OpenCV. ActivMedia Color Tracking System(ACTS), is mainly used for blob tracking where multiple images of the blob to be tracked is supplied to the camera as part of the training process. After which the robot can be made to follow the blob. If the blob is spherical, the radius can be used as a variable to push the robot forward or backward and the center of the sphere can be used to pan and tilt the camera accordingly. The details about using the ACTS Software can be found in the ACTS User Manual [9]. Another great resource is OpenCV which is excellent for vision processing tasks. Interested readers can visit [7] to learn more about OpenCV or read the book "Learning OpenCV: Computer vision with the OpenCV library" [10] by Gary Bradski.

6. USING THE SIMULATOR

A useful feature of the ARIA library is that it comes with an easy to setup simulator, called MobileSim. Beginners can safely test their programs on a simulator before trying them out on the real robot. An introductory robotics course can benefit twofold. The programs in such courses are often complex and a simulator prevents damage to the robot and allows for resolving any ensuing bugs and program logic before actual testing. The other benefit is that the ARIA class of robots are expensive, thus it makes them a limiting resource. The simulator can also override this problem.

The link provided in Chapter 2 "Getting Started" enlists MobileSim as one of the downloadable packages. In Linux, once the package is installed using the `dpkg -i` command, pressing ALT+F2 and typing `MobileSim` or typing the same on a shell prompt, starts the simulator. This opens a dialog box prompting for a robot model and a map. A more explicit command can be typed by including the robot model and the location of the map which prevents the initial dialog box to open:

```
MobileSim    -m    myPath/myMap.map    -r
myRobotModel
```

Once the Simulator is running, the source code can be executed using the `./myProgram` on the shell prompt. If need be, ARIA also comes with a Map editing program, Mapper3Basic. The program allows for setting up new environments and modifying existing ones using an easy-to-use interfaces.

7. USING THE WRAPPERS

ARIA provides two wrappers for developers who would like to program in Java or Python. In essence, the wrappers supply programmers with a Java and Python API that actually calls the standard C++ library (libAria.so on Linux and Aria.dll on Windows) using the wrapper layer[5]. However the ARIA library has not been thoroughly tested using the wrapper packages. Unlike the huge C++ code repository of examples and test programs in the ARIA package, the Java and Python package installation does not provide ample code repository. However, it is hoped that with increasing use of the library in those languages, developers will share their valuable source code and create proper Java and Python documentation for the ARIA library.

7.1 Java Wrapper

A Java SDK version 1.6 (Java SE6) or higher must be installed prior to the use of the wrapper. Unlike the installation of the Python wrapper, installation of the Java wrapper will not complain if a Java SDK is not installed. In Windows it is necessary to set the bin folder of ARIA in the list of environment path variable. This allows Java programs to access the Java wrapper's dynamic linked library found in the bin folder. As described in Section 5, "Using the Simulator", Java programs too can be run using the simulator. A simulator needs to be started before running the compiled Java programs. To compile the programs in Windows or Linux use the following command, assuming the program is located in the examples directory:

```
javac -classpath ../java/Aria.jar
Hello.java
```
OR

```
javac -cp ../java/Aria.jar Hello.java
```

There is a slight difference between the way the program is run on Windows and Linux. In Windows use the following command:

```
java -cp ../java/Aria.jar;. Hello
```

In Linux the semicolon is replaced by the colon, that is:

```
java -cp ../java/Aria.jar:. Hello
```

See Appendix 1.10 for a sample Java program that does simple motor control obtained from one of the downloaded examples, provided for easy reference.

7.2 Python Wrapper

Before installing the Python wrapper package, one should install a version of Python that the wrapper was built with. The wrapper found at the Downloads section requires Python version 2.4 for Debian Linux or Windows and Python version 2.2 for RedHat [5]. If a different version is available on the host machine then the wrappers need to be rebuilt. If Python 2.6 is used, a renaming of the dynamic linked library (`_AriaPy.dll` or `_AriaPy.so` to `_AriaPy.pyd`) is required under the python directory. An environment path variable needs to be set up that links to the python folder under ARIA. Similarly, a simulator can be used to test the Python programs. Again, just as decribed in Section 5, a simulator needs to be started before running a python program. Double clicking the file on Windows will run the python program or typing the following command on Windows or Linux will run the program:

```
python myPythonExample.py
```

See Appendix 1.11 for a sample Python program that does simple motor control obtained from one of the downloaded examples, provided for easy reference.

8. CONCLUSION

The main motivation of this paper is to create a guide that breaks this initial block both for beginners and researchers and allow for maximum adoption of this great library. The library is also ideal for an introductory robotics class. The Java support of the library provided by the Java wrapper, can be used to teach introductory Object Oriented programming to high school students. Such a course would have a visual component to it. ARIA's library is highly featured and one cannot miss the edutainment part it would provide to the students. As such, the paper can be used as a lab handout for a course in Robotics or one designed to teach Object Oriented Principles (OOP) as an initiative towards tactile or kinesthetic learning.

9. REFERENCES

[1] Borgström, "ARIA and Matlab Integration With Applications," 2005.

[2] K.J. O'Hara and J.S. Kay, "Investigating open source software and educational robotics," Journal of Computing Sciences in Colleges, vol. 18, 2003.

[3] A. Farinelli, G. Grisetti, and L. Iocchi, "Design and implementation of modular software for programming mobile robots," International Journal of Advanced Robotic Systems, vol. 3, 2006.

[4] OpenSSH. (2010, June 7). In *Wikipedia, The Free Encyclopedia*. Retrieved 19:06, June 13, 2010, from http://en.wikipedia.org/w/index.php?title=OpenSSH&oldid=366479723

[5] "MobileRobots Advanced Robotics Interface for Applications (ARIA) Developer's Reference Manual 2.7.2," ActivMedia Robotics, LLC, MobileRobots Inc, 2009.

[6] "Linux UVC Documentation," Linux UVC Open-Facts, berliOS, 05:29 Nov 15, 2007. [html]. http://openfacts.berlios.de/index-en.phtml?title=Linux+UVC [Accessed: May 20, 2010]

[7] "How to use OpenCV to capture and display images from a camera," CameraCapture OpenCV Wiki, OpenCV, 2008-05-28 14:02:33. [html]. http://opencv.willowgarage.com/wiki/CameraCapture [Accessed: May 20, 2010]

[8] Laser Range-Finder Installation and Operations Manual, Version 1, ActivMedia, NH, USA, (2002)

[9] ACTS User Manual, Version 6, ActivMedia, NH, USA, (2006)

[10] G. Bradski and A. Kaehler, Learning OpenCV: Computer vision with the OpenCV library, O'Reilly Media, Inc., 2008.

[11] Monisit, Kristofer . "UVC". Ubuntu. May 20th 2010 <https://help.ubuntu.com/community/UVC>.

[12] ROS (Robot Operating System). (2010, June 11). In *Wikipedia, The Free Encyclopedia*. Retrieved 20:14, June 13, 2010, from http://en.wikipedia.org/w/index.php?title=ROS_(Robot_Operating_System)&oldid=367349467

[13] A. Robotics, "Pioneer 2/PeopleBot Operations Manual, ActivMedia Robotics, 44 Concord St., Peterborough NH, 03458," October, vol. 4, 2001.

APPENDIX 1

1. CREATING A MAKE FILE

Create a file and name it **Makefile**. Say myProgram.cpp is the name of the C++ source code and myProgram will be the name of the executable after compilation, then insert the following lines in the makefile:

```
all: program

CFLAGS=-fPIC -g -Wall
ARIA_INCLUDE=-I/usr/local/Aria/include
ARIA_LINK=-L/usr/local/Aria/lib -lAria -lpthread -ldl -lrt /usr/lib/libstdc++.so.5

%: %.cpp
	$(CXX) $(CFLAGS) $(ARIA_INCLUDE) $< -o $@ $(ARIA_LINK)
```

Save this file in the same location as the source file. Now, to compile the source code, type **`make myProgram.cpp myProgram`** at the command line. To execute the code, type **`./myProgram`**. It should be noted that using the existing Makefile that comes with the installation takes a longer time to compile, however typing just make withouth any arguments using the default makefile will compile all the programs in the examples folder, creating the respective executables for each program.

2. STEPS TO COMPILE ARIA PROGRAMS IN VISUAL STUDIO 2008:

1. Download Visual C++ Express if any version of Visual Studio is not installed.
2. Create a folder (say in C: drive), and name it Aria. Within this folder, create two subfolders; “include” and “lib”.
3. Inside the “include” folder move all the header files that came with the installation of ARIA.
4. Move all the “dll” files inside the “lib” folder.
5. When creating a new project, under Tools->Options->Projects and Solutions->VC++ Directories, add the above paths for the two folders in their respective directory settings.
6. Add the same paths (in this case it would be: “C:/Aria/lib” and “C:/Aria/include”) to your Environment Variables.

3. COMMUNICATING WITH THE ROBOT [5]

```
#include "Aria.h"

/** @example myDemo.cpp
 */

int main(int argc, char** argv)
{
  // mandatory init
  Aria::init();

  // Declarations for basic startup:
  // [1] set up our parser
  ArArgumentParser parser(&argc, argv);
  // [2] set up our simple connector that takes in the parser object
  ArSimpleConnector simpleConnector(&parser);
  // [3] main robot class
  ArRobot robot;

  // loads the default arguments if no arguments were passed
  parser.loadDefaultArguments();
  // parse the command line... fail and print the help if the parsing fails
  // or if the help was requested
  if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
  {
    Aria::logOptions();
    exit(1);
  }


  // set up the robot for connecting
  if (!simpleConnector.connectRobot(&robot))
  {
    printf("Could not connect to robot... exiting\n");
    Aria::exit(1);
  }

  // true means if we lose connection the robot should stop
  robot.runAsync(true);

  // Connection has been established with the robot now
  // the lock() and unlock() commands ensure no interference with other commands
  // and appear in a block
  robot.lock(); // every time, we send a command to robot, we lock it
  robot.comInt(ArCommands::ENABLE, 1);        // enable the motors
  robot.unlock(); // and unlock it after it has been executed

  // Command to turn robot 90 degrees to the right
  robot.lock();
  robot.setHeading(90);
  robot.unlock();

  // shutdown and getout
  Aria::shutdown();
}
```

4. COMMUNICATING WITH THE SENSORS: LASER, BUMPERS, SONAR & CAMERA

```
#include "Aria.h"

/** @example mySensor.cpp
 */

int main(int argc, char** argv)
{
  Aria::init();
  ArArgumentParser parser(&argc, argv);
  ArRobot robot;

  // Initializing all the devices:
  // [1] Laser:
  ArSick myLaser;
  // [2] Sonar:
  ArSonarDevice mySonar;
  // [3] Bumpers:
  ArBumpers bumpers;
  // [4] Camera:
  ArVCC4 myCamera(&robot);
  myCamera.init();


  // Add the sensors to the robot
  robot.addRangeDevice(&mySonar);
  robot.addRangeDevice(&myBumpers);

  myLaser.runAsync();
  if (!connector.connectLaser(&myLaser))
  {
    exit(0);
  }
  robot.addRangeDevice(&myLaser);

  parser.loadDefaultArguments();
  if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
  {
    Aria::logOptions();
    exit(1);
  }


  if (!simpleConnector.connectRobot(&robot))
  {
    printf("Could not connect to robot... exiting\n");
    Aria::exit(1);
  }
  robot.runAsync(true);
  robot.lock();
  robot.comInt(ArCommands::ENABLE, 1);
  robot.unlock();

  // Command to turn robot 90 degrees to the right
  robot.lock();
  robot.setHeading(90);
  robot.unlock();
  Aria::shutdown();
}
```

5. KEY METHODS TO CONTROL ROBOT’s MOTION [5]

Method	Details
double getAbsoluteMaxTransVel()	Returns the robot’s maximum translational velocity (mm/s)
double getAbsoluteMaxRotVel()	Returns the robot’s maximum rotational velocity (°/s)
bool setAbsoluteMaxTransVel(double maxVel)	Sets the maximum translational velocity (mm/s). It returns true if it can set the value provided, otherwise it returns false.
bool setAbsoluteMaxRotVel(double maxVel)	Sets the maximum rotational velocity (°/s). It returns true if it can set the value provided, otherwise it returns false.
void setVel(double velocity)	Sets the translational velocity (mm/s)

void setRotVel(double velocity)	Sets the rotational velocity (°/s)
void setVel2(double leftVelocity, double rightVelocity)	Sets the desired velocity of the left wheel and the right wheel of the robot; both in mm/s
void stop()	Stops the robot
void move(double distance)	Moves the robot in stated distance either forward or backward based on sign of value.
void setHeading(double heading)	Sets the absolute heading of the robot in degrees
Void setDeltaHeading(double deltaHeading)	Sets the new heading to provided value relative to the one it had before the method call

6. PROGRAMMATIC ACCESS OF SONAR VALUES [4]

1. A slice of the polar region (front: -90 to 90)

```
#include "Aria.h"

/** @example mySonar1.cpp
 */

int main(int argc, char** argv)
{
  Aria::init();
  ArArgumentParser parser(&argc, argv);
  ArRobot robot;

  ArSonarDevice mySonar;
  robot.addRangeDevice(&mySonar);

  double value;  // variable to hold the closest value from all the sonar readings

  // angleAtValue is passed as pointer to method to retrieve angle at closest value
  double angleAtValue;

  value=mySonar.currentReadingPolar(-90,90, &angleAtValue);
  …
}
```

2. Specific Sonar array

```
/** @example mySonar2.cpp
 */

int main(int argc, char** argv)
{
  Aria::init();
  ArArgumentParser parser(&argc, argv);
  ArRobot robot;

  ArSonarDevice mySonar;
  robot.addRangeDevice(&mySonar);

  ArSonarDevice mySonar;
  robot.addRangeDevice(&mySonar);
  int id = 5;// say, the specific sonar array is 5

  ArSensorReading* values; // This class abstracts range and angle read from sonar

  value = robot->getSonarReading(5);
  double range = value->getRange();
  double angle = value->getSensorTh();
  …
}
```

3. **All the Sonar sensors**

```
/** @example mySonar3.cpp
 */

int main(int argc, char** argv)
{
  Aria::init();
  ArArgumentParser parser(&argc, argv);
  ArRobot robot;

  ArSonarDevice mySonar;
  robot.addRangeDevice(&mySonar);

  int total = robot->getNumSonar(); // get the total number of sonar on the robot

  ArSensorReading* values; // This class abstracts range and angle read from sonar

  for( int i = 0; i < total; i++ )
  {
    value = robot->getSonarReading(i);
    double range = value->getRange();
    double angle = value->getSensorTh();
  }
  …
}
```

7. PROGRAMMATIC ACCESS OF LASER VALUES [4]

1. **Slice of the polar region (front: -90 to 90)**

```
#include "Aria.h"

/** @example myLaser1.cpp
 */

int main(int argc, char** argv)
{
  Aria::init();
  ArArgumentParser parser(&argc, argv);
  ArRobot robot;

  ArSick myLaser;
  robot.addRangeDevice(&myLaser);

  double value;  // variable to hold the closest value from all the laser readings
  // angleAtValue is passed as pointer to method to retrieve angle at closest value
  double angleAtValue;

  value=myLaser.currentReadingPolar(-90,90, &angleAtValue);
  …
}
```

2. **All the laser values**

```
#include "Aria.h"

/** @example myLaser2.cpp
 */
using namespace std;
int main(int argc, char** argv)
{
  Aria::init();
  ArArgumentParser parser(&argc, argv);
  ArRobot robot;

  ArSick myLaser;
  robot.addRangeDevice(&myLaser);

  // Create a list of ArSensorReading values
  list<ArSensorReading *> *values;

  // Create an iterator to loop through the above values
  list<ArSensorReading *>::const_iterator it;

  values = myLaser->getRawReadings();
  int i = -1;
  double angle, range;

  for( it = values->begin(); it != values->end(); it++ )
  {
    i++;
    range = (*it)->getRange();
    angle = (*it)->getSensorTh();
  }
  …
}
```

8. PROGRAMMATIC CONTROL OF CAMERA's PAN-TILT BASE[5]

Method	Details
double getMaxNegPan()	Returns the minimum angle(°) that the camera can pan to
double getMaxPosPan()	Returns the maximum angle(°) that the camera can pan to
double getMaxNegTilt()	Returns the minimum angle(°) that the camera can tilt to
double getMaxPosTilt()	Returns the maximum angle(°) that the camera can tilt to
int getMinZoom()	Returns the minimum attainable zoom by the camera
int getManZoom()	Returns the maximum attainable zoom by the camera

double getPan()	Returns the current pan angle(°)
double getTilt()	Returns the current tilt angle(°)
int getZoom()	Returns the current zoom state of the camera
bool haltPanTilt()	Halts all pan tilt movement and returns and returns true on success and false if unable to halt
bool haltZoom()	Halts zoom movement
bool pan(double degree)	Pans the camera to the angle argument(°), the returned Boolean value indicates success or failure
bool panRel(double degree)	Pans the camera to an angle relative to current pan angle
bool tilt(double degree)	Tilts the camera to the angle argument(°), the Boolean flag indicates success or failure
bool tiltRel(double degree)	Tilts the camera to an angle relative to current tilt angle
bool pantilt(double pdeg, double tdeg)	Pans and Tilts the camera to pdeg and tdeg respectively
bool pantiltRel(double pdeg, double tdeg)	Pans and Tilts the camera to pdeg and tdeg which are angles relative to the current pan and tilt angle
bool zoom(int value)	Zooms the camera to the integer value, the returned Boolean value indicates success or failure

9. ACCESSING CAMERA IMAGES

1. Linux: Using UVC Drivers (V4L)

a. Header file:

```
/// @file Image.h
/// @brief Header generic image class

#ifndef __IMAGE_H
#define __IMAGE_H

class Image
{
public:
        /// @brief Constructor
        /// @param[in] w Width of the image
        /// @param[in] h Height of the image
        /// @param[in] ptr Raw image bytes
        /// @param[in] sz Size in bytes of the given image
        Image(int w, int h, char* ptr, int sz) : width(w), height(h), data(ptr),
size(sz) { }

        /// @brief Destructor, removes image data
        ~Image() { delete[] data; }

        char* getRawData() { return data; }
        int getWidth() const { return width; }
        int getHeight() const { return height; }
        int getSize() const { return size; }

        void setRawData(char* ptr) { data = ptr; }
        void setWidth(int w) { width = w; }
        void setHeight(int h) { height = h; }
        void setSize(int sz) { size = sz; }


private:
        int width, height, size;
        char* data;
};

#endif
```

```
@file Camera.cpp
/// @file Camera.h
/// @brief Header for Camera class object

#ifndef __CAMERA_H
#define __CAMERA_H

#include "Image.h"
#include <stdlib.h>
#include <pthread.h>

/// @brief A structure describing a frame's buffer
struct Framebuffer
{
        void* addr; //< The memory-mapped address of the buffer
        int length; //< The length of the buffer
};

class Camera
{
private:
        int cameraID; ///< Unique camera identifier
        int camfile; ///< UNIX file handle for camera

        Framebuffer* buffers; //< Array of frame buffers
        int nr_buffers; //< Number of frame buffers

        //Multithreaded stuff
        pthread_t thread; ///< Handle to thread that controls this camera
        volatile Image* last_captured; //< Pointer to last image captured. If NULL,
then the thread will capture a new image
        volatile bool active;

        bool initCamera();
public:
        Camera() : cameraID(-1), camfile(-1), buffers(NULL), nr_buffers(0) { }
        ~Camera();

        int getCameraFile() const { return camfile; }
        Framebuffer* getFramebuffer(int i) { return &buffers[i]; }

        volatile Image* getLastFrame() const { return last_captured; }
        void setLastFrame(Image* img) { last_captured = img; }
        int isActive() { return active; }

        /// @brief Initializes the camera
        /// @param id Camera ID that uniquely identifies the camera
        /// @return The error state, one of HardwareError codes
        HardwareError initialize(int id);

        /// @brief Begins capture of an image from the camera
        void beginCaptureImage();

        /// @brief Releases the capture device housekeeping
        void releaseCapture();
};
#endif
```

b. **Source file:**

```
/// @file Camera.cpp
/// @brief Code for Camera class object

//This uses Video For Linux 2.0 (V4L2) API

#include "Camera.h"
#include <stdlib.h> // NULL defintion
#include <stdio.h> // sprintf()
#include <errno.h> // errno
#include <string.h> // memset()
#include <fcntl.h> // fcntl()
#include <unistd.h> // open(), close(), etc.
#include <sys/mman.h> // mmap()
#include <sys/ioctl.h> // ioctl()
#include <asm/types.h> //required for videodev2.h
#include <linux/videodev2.h>
#include <pthread.h>
static int open_cam(const char* name);
static int waitioctl(int fd, int request, void* arg);
static void stream_on(int fd);
static void stream_off(int fd);

static void disp_errno(const char* str);

static void* camera_thread(void* cam);

Camera::~Camera()
{
}

HardwareError Camera::initialize( int CameraID )
{
        char charname[128];

        //Ignore double initialize() calls
        if(this->isActive())
                return HWERR_SUCCESS;
        sprintf(charname, "/dev/video%d", CameraID);

        //Open camera device, report any errors
        camfile = open_cam(charname);
        if(camfile == -1)
                return HWERR_DISCONNECTED;
        else if(camfile == -2)
                return HWERR_UNABLE_TO_INITIALIZE;

        if(this->initCamera() == false)
        {
                close(camfile);
                return HWERR_UNABLE_TO_INITIALIZE;
        }

        //Set up thread variables
        last_captured = (Image*)0xdeadc0de;
        active = true;
        if(pthread_create(&this->thread, NULL, camera_thread, (void*) this) != 0)
        {
                close(camfile);
                return HWERR_UNABLE_TO_INITIALIZE;
        }

        return HWERR_SUCCESS;
}


void Camera::beginCaptureImage()
{
        //Signal to thread to capture the image
        last_captured = NULL;
}

void Camera::releaseCapture()
{
        close(camfile);
        active = false;
}

static int waitioctl(int fd, int request, void* arg)
{
        int retval;

        do
        {
                retval = ioctl(fd, request, arg);
        } while(retval == -1 && EINTR == errno);

        return retval;
}

static int open_cam(const char* name)
{
        struct stat st;
        int fd;
        //Can't stat() the file? (doesn't exist)
        if(stat(name, &st) == -1)
                return -1;

        //Open camera for R/W access and make non-blocking
        fd = open(name, O_RDWR | O_NONBLOCK, 0);

        //Can't open file?
        if(fd == -1)
                return -2;

        return fd;
}

bool Camera::initCamera()
{
        struct v4l2_capability cap;
        struct v4l2_format fmt;
        struct v4l2_requestbuffers req;
        int fd = this->camfile;
        int i;

        //Query hardware capabilities
        if(waitioctl(fd, VIDIOC_QUERYCAP, &cap) == -1)
                return false;

        //Does device support streaming?
        if(cap.capabilities & V4L2_CAP_STREAMING == 0)
                return false;

        //Set up camera format
        memset(&fmt, 0, sizeof(fmt));
        fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        fmt.fmt.pix.width = CAMERA_WIDTH;
        fmt.fmt.pix.height = CAMERA_HEIGHT;
        fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
        fmt.fmt.pix.field = V4L2_FIELD_NONE;

        //Tell V4L to set camera to use this format
        if(waitioctl(fd, VIDIOC_S_FMT, &fmt) == -1)
                return false;

        //Request buffers
        memset(&req, 0, sizeof(req));
        req.count = 4;
        req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        req.memory = V4L2_MEMORY_MMAP;
        if(waitioctl(fd, VIDIOC_REQBUFS, &req) == -1)
                return false;

        // Really need at least double buffering
        if(req.count < 2)
                false;

        buffers = new Framebuffer[req.count];

        //Memory map each buffer
        for(i=0; i<req.count; i++)
        {
                struct v4l2_buffer buf;
                memset(&buf, 0, sizeof(buf));

                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory = V4L2_MEMORY_MMAP;
                buf.index = i;

                //Query the i'th buffer's information
                if(waitioctl(fd, VIDIOC_QUERYBUF, &buf))
                {
                        delete[] buffers;
                        buffers = NULL;
                        return false;
                }

                //Set up the memory map
                buffers[i].length = buf.length;
                buffers[i].addr = mmap(NULL, buf.length, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, buf.m.offset);

                if(MAP_FAILED == buffers[i].addr)
                {
                        delete[] buffers;
                        buffers = NULL;
                        return false;
                }

                //Now, enqueue the buffer
                if(waitioctl(fd, VIDIOC_QBUF, &buf) == -1)
                {
                        delete[] buffers;
                        buffers = NULL;
                        return false;
                }

        }
        stream_on(fd);
        return true;
}

static void stream_on(int fd)
{
        enum v4l2_buf_type type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

        waitioctl(fd, VIDIOC_STREAMON, &type);
}

static void stream_off(int fd)
{
        enum v4l2_buf_type type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

        waitioctl(fd, VIDIOC_STREAMOFF, &type);
}
```

```
//////////////////////////////////////////
// CAMERA THREAD
//////////////////////////////////////////
static void* camera_thread(void* cam)
{
        Camera* camera = (Camera*)cam;
        int fd = camera->getCameraFile();

        while(camera->isActive())
        {
                if(camera->getLastFrame() != NULL)
                        usleep(1000); //sleep for 1000 usec, or 1 msec
                else
                {
                        fd_set fds;

                        //Set up an unlimited wait-for-read fd set
                        FD_ZERO(&fds);
                        FD_SET(fd, &fds);

                        //Wait for an image...
                        if(select(fd+1, &fds, NULL, NULL, NULL) > 0)
                        {
                                struct v4l2_buffer buf;
                                //Dequeue the buffer
                                memset(&buf, 0, sizeof(buf));
                                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                                buf.memory = V4L2_MEMORY_MMAP;
                                if(waitioctl(fd, VIDIOC_DQBUF, &buf) != -1)
                                {
                                        //Create an in-memory duplicate of this
information
                                        char* rawdata = new char[camera-
>getFramebuffer(buf.index)->length];
                                        memcpy(rawdata, camera-
>getFramebuffer(buf.index)->addr, camera->getFramebuffer(buf.index)->length);

                                        //Create a new image from this
                                        Image* img = new Image(CAMERA_WIDTH,
CAMERA_HEIGHT, rawdata, camera->getFramebuffer(buf.index)->length);

                                        //Enqueue the buffer we just dequeued,
invalidate all other buffers (so we don't read old data next time)
                                        waitioctl(fd, VIDIOC_QBUF, &buf);
                                        //stream_off(fd);

                                        //Post frame to other thread
                                        camera->setLastFrame(img);
                                }

                        } //select()
                } // need frame
        } // while thread is active

        return NULL;
}
```

```
static void disp_errno(const char* str)
{
        const char* reason = "(unknown)";
        switch(errno)
        {
                case EIO: reason = "EIO"; break;
                case EINTR: reason = "EINTR"; break;
                case EINVAL: reason = "EINVAL"; break;
        }
        printf("%s: %s\n", str, reason);
}
```

c. **Test program:**

```
/// @brief Camera capture image test
#include "Camera.h"

#include <stdio.h>

void testMain()
{
        Camera cam;
        Image* img;
        int i;
        if(cam.initialize(0) != HWERR_SUCCESS)
        {
                Printf("Failed to initialize camera /dev/video0");
                return;
        }
        cam.releaseCapture();
        return;
}
```

2. **Windows using OpenCV: [7]**

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>

// A Simple Camera Capture Framework
int main() {

  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if( !capture ) {
    fprintf( stderr, "ERROR: capture is NULL \n" );
    getchar();
    return -1;
  }

  // Create a window in which the captured images will be presented
  cvNamedWindow( "mywindow", CV_WINDOW_AUTOSIZE );

  // Show the image captured from the camera in the window and repeat
  while( 1 ) {
    // Get one frame
    IplImage* frame = cvQueryFrame( capture );
    if( !frame ) {
      fprintf( stderr, "ERROR: frame is null...\n" );
      getchar();
      break;
    }

    cvShowImage( "mywindow", frame );
    // Do not release the frame!

    //If ESC key pressed, Key=0x10001B under OpenCV 0.9.7(linux version),
    //remove higher bits using AND operator
    if( (cvWaitKey(10) & 255) == 27 ) break;
  }

  // Release the capture device housekeeping
  cvReleaseCapture( &capture );
  cvDestroyWindow( "mywindow" );
  return 0;
}
```

10. **Java Example: Direct Drive commands [5]**

```
/* A simple example of connecting to and driving the robot with direct

 * motion commands. */
import com.mobilerobots.Aria.*;

public class simple
{
  static
  {
    try
    {
        System.loadLibrary("AriaJava");
    }
    catch (UnsatisfiedLinkError e)
    {

      System.err.println("Native code library libAriaJava failed to load. Make sure
that its directory is in your library path\n" + e);
      System.exit(1);

    }
  }
  public static void main(String argv[])
  {
    System.out.println("Starting Java Test");

    Aria.init();

    ArRobot robot = new ArRobot();
    ArSimpleConnector conn = new ArSimpleConnector(argv);

    if(!Aria.parseArgs())
    {
      Aria.logOptions();
      Aria.shutdown();
      System.exit(1);
    }

    if (!conn.connectRobot(robot))
    {
      System.err.println("Could not connect to robot, exiting.\n");
      System.exit(1);
    }
    robot.runAsync(true);
    robot.lock();
    System.out.println("Sending command to move forward 1 meter...");
    robot.enableMotors();
    robot.move(1000);
    robot.unlock();
    System.out.println("Sleeping for 5 seconds...");
    ArUtil.sleep(5000);
    robot.lock();
    System.out.println("Sending command to rotate 90 degrees...");
    robot.setHeading(90);
    robot.unlock();
    System.out.println("Sleeping for 5 seconds...");
    ArUtil.sleep(5000)
    robot.lock();
    System.out.println("Robot coords (" + robot.getX() + ", " + robot.getY() +
        ", " + robot.getTh() + ")");
    robot.unlock();

    robot.lock();
    System.out.println("exiting.");
    robot.stopRunning(true);
    robot.unlock();
    robot.lock();
    robot.disconnect();
    robot.unlock();
    Aria.shutdown();
  }

}
```

11. **Python Example: Direct Drive commands [5]**

```
import com.mobilerobots.Aria.*;
from AriaPy import *import sys

# Global library initialization, just like the C++ API:
Aria.init()

# Create a robot object:
robot = ArRobot()

# Create a "simple connector" object and connect to either the simulator
# or the robot. Unlike the C++ API which takes int and char* pointers,
# the Python constructor just takes argv as a list.
print "Connecting..."

con = ArSimpleConnector(sys.argv)
```

```
if not con.parseArgs():
    con.logOptions()
    Aria.exit(1)

if not con.connectRobot(robot):
    print "Could not connect to robot, exiting"
    Aria.exit(1)

# Run the robot threads in the background:
print "Running..."
robot.runAsync(1)

# Drive the robot a bit, then exit.
robot.lock()

print "Robot position using ArRobot accessor methods: (", robot.getX(), ",",
robot.getY(), ",", robot.getTh(), ")"

pose = robot.getPose()
print "Robot position by printing ArPose object: ", pose
print "Robot position using special python-only ArPose members: (", pose.x, ",",
pose.y, ",", pose.th, ")"
print "Sending command to move forward 1 meter..."
robot.enableMotors()
robot.move(1000)
robot.unlock()
print "Sleeping for 5 seconds..."
ArUtil.sleep(5000)
robot.lock()
print "Sending command to rotate 90 degrees..."
robot.setHeading(90)
robot.unlock()
print "Sleeping for 5 seconds..."
ArUtil.sleep(5000)
robot.lock()
print "Robot position (", robot.getX(), ",", robot.getY(), ",", robot.getTh(), ")"
pose = robot.getPose()
print "Robot position by printing ArPose object: ", pose
print "Robot position using special python-only ArPose members: (", pose.x, ",",
pose.y, ",", pose.th, ")"
robot.unlock()
print "Exiting."
Aria.shutdown()
```