NOTES

Table of Contents

[Testing 2](#_Toc403551760)

[CoolTerm 2](#_Toc403551761)

[Test Cases 3](#_Toc403551762)

[Move (0x01) Forward Half Speed Test 3](#_Toc403551763)

[Camera Home (0x04) Test 3](#_Toc403551764)

[EEPROM Write (0x07) Test 4](#_Toc403551765)

[Appendix Arduino MEGA ADK Support 4](#_Toc403551766)

[How to Update Adb Library to Support Arduino version 1.0+ 4](#_Toc403551767)

Revision History

Version 8.1

1. Although it should never occur, out-of-range test updated to follow FSM model.

command file/command\_echo function, and command file/throw\_error function

1. Unlike the Arduino UNO the Leonardo (ATmega32U4) allows data to be transmitted through USART and USB peripheral subsystems. If telemetry is sent over Bluetooth (USART), then redundant command echo and error messages are sent as text using the USB port to the Arduino IDE Serial Monitor.
2. Watchdog timer added

# Getting Started

You can download the arxrobot\_firmware from the Arxterra GitHub releases page: <https://github.com/arxterra/arxrobot-firmware/releases>.

Near the top of the arxrobot\_firmware tab you will see a sequence of definitions.

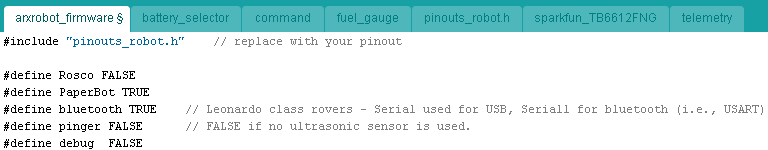


Figure 0.1 Configuring the code for a PaperBot robot

Based on the Arxterra robot you have and the features included, set these fields accordingly. The setting for Figure 0.1 correspond to a standard PaperBot. Figure 0.2 show the settings for a mini-Rosco rover. The mini-Rosco uses bluetooth communictions versus a standard Rosco which used USB, and this case the mini-Rosco is equipped with an ultrasonic pinger.

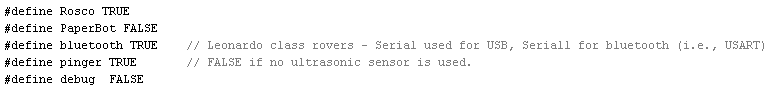


Figure 0.2 Configuring the code for a mini-Rosco

Once you have configured your code. Open the pinouts\_robot.h tab and make sure your rover’s pinouts are correct. You can update the table as needed.

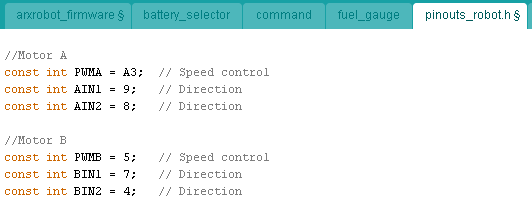


Figure 0.3 Defining Arduino-to-robot pinouts

The remainder of this document covers how to modify the arxrobot\_firmware for your robot.

# Commands

Command and telemetry data are sent as byte arrays formatted into packets. The format of a command packet is shown in Figure 1.0. Each command packet consists of the following bytes.

1. The Packet ID is an Arxterra unique byte indicating the start of a packet. For commands, this byte by definition is hexadecimal A5.
2. The Packet Length is the number of bytes following minus the LRC byte.
3. The Instruction ID is an 8-bit code identifying the instruction to be executed.
4. Depending on the instruction, the instruction ID byte is followed by some number of bytes.
5. The Longitudinal Redundancy Check (LRC) byte provides a simple parity check of the command packet. The LRC is generated over the entire packet.

### 

Figure 1.0 Command Packet Format

## Move (0x01) Forward Half Speed Test Example

In this example the robot is commanded to proceed forward at half speed (50% duty cycle).

**i = 0 1 2 3 4 5 6 7**

**N = 1 2 3 4 5**

**A5 05 01 01 80 01 80 A1 MOV forward half speed**

sendPacket

data[0] A5 = Command Packet ID

data[1] 05 = Packet Length N w/o parity byte

data[2] 01 = Move ID

data[3] 01 = Left Motor Forward

data[4] 80 = Half Speed (128/255)

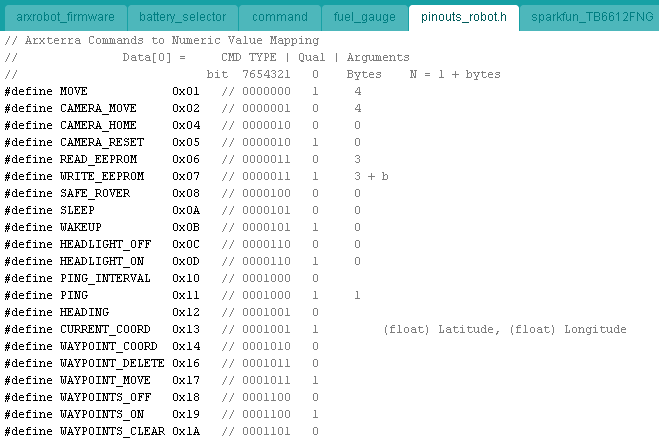
data[5] 01 = Right Motor Forward

data[6] 80 = Half Speed (128/255)

data[7] A1 = Longitudinal Redundancy Check (LRC)

## Robot Commands

The latest table of command definitions is included in the pinouts\_robot.h file. Within the Arduino IDE these command definitions may be found under the pinouts\_robot.h tab. Figure 2.0 shows the command definitions for arxrobot-firmware alpha version 0.8.2.

Figure 2.0 Commands to Numeric Value Map

Each command is comprised of a 7-bit command type field, followed by a 1-bit qualifier. The qualifier allows 1-byte commands with a single Boolean argument. For example Headlight ON/OFF (0b0000110:1/0).

## Custom Commands

The Arxterra control panel allows you to create your own custom commands based on the unique requirements of your robot. Figure 3.0 shows a control panel designed to support a Hexapod robot.

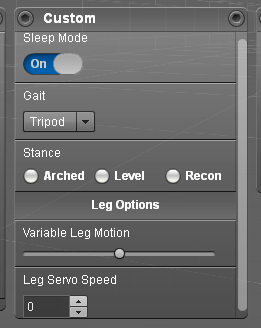


Figure 3.0 Custom Control panel for a Hexapod Robot

Control Panel User interface (UI) design widgets include radio buttons, sliders, drop down menus, and more.

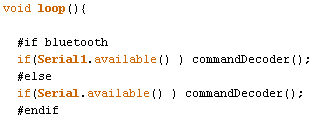
The Arxterra Arduino program reserves command IDs **0x40** to **0x5F** for implementing these user defined commands on the robot.

To learn how to create your own custom UI widget, read “[Creating Custom Commands With the Arxterra Application](http://arxterra.com/creating-custom-commands-with-the-arxterra-application/)” by Tommy Sanchez.

### Implementing Commands

#### How the Arxrobot firmware detects, reads, and handles commands

All Arduino programs contain a setup() and a loop() method. Commands are detected in the loop() method.



Once detected, commands are decoded in the commandDecoder() function. You can find the command decoder code under the command tab within the Arduino IDE (Integrated Development Environment).

 The commandDecoder method implements a Moore FSM which reads the incoming byte array (see Figure 1.0). If the command is successfully decoded then the commandHandler function is called from state 3 of the FSM.



The commandHandler function definition includes parameters data and N. The data parameter points to the received byte-array. The N parameter is the length of the packet as previously defined (see Figure 1 and associated definitions). It is simplest to understand these two parameters by again looking at the move (0x01) command example.

**i = 0 1 2 3 4 5 6 7**

**N = 1 2 3 4 5**

**Data = A5 05 01 01 80 01 80 A1 MOV forward half speed**

The first byte A5 identifies this as the start of an Arxterra command packet. The second byte 05 is the value sent as the N argument. The third byte (data[2]) is the Arxterra command, in this case move (0x01). One of the first things the command handler function does is to assign this third byte to the unsigned byte variable cmd.

uint8\_t cmd = data[2];

Shortly following this assignment statement is a series of if-else if statements which allow direct action to be taken by the Arduino based on the command received.

if (cmd == MOVE) {

...

}

else if (cmd == CAMERA\_MOVE){

...

}

else if (cmd == CAMERA\_RESET){ ...

The next section will look at how to adapt the existing code, within each command, to the specific needs of your robot.

#### How to integrate the Arduino code for your robot’s unique command(s)

IMPORTANT

Before you integrate your robot’s unique functions into the arxrobot application, begin by developing and testing your new functions in a separate Arduino ino file(s). Once you know they work you can add them to the arxrobot\_firmware code, the subject of this section

You can take two approaches to modifying the code for your robot. The simplest is to throw away the code within the else if block and adding your own. The second is to add your code in such a way that the original functionality of the code is not lost. Here is how to accomplish the second option.

Going back to the arxrobot\_firmware tab (see Introductions Section) you will see a sequence of definitions.

#define PaperBot TRUE

#define Rosco FALSE

#define Pathfinder FALSE

#define bluetooth TRUE

#define pinger FALSE // FALSE if no ultrasonic sensor is used.

#define debug TRUE

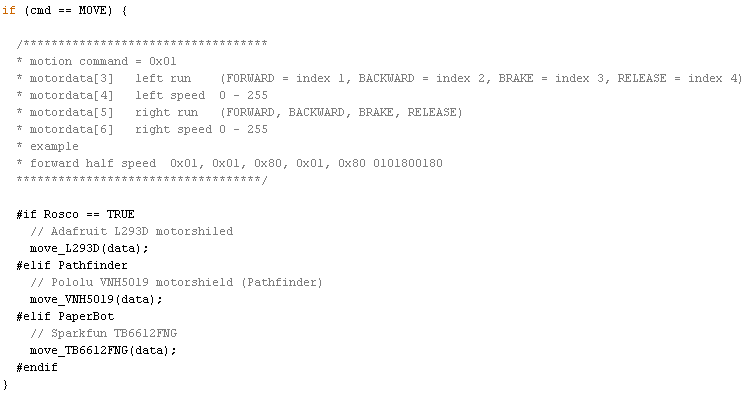
The #define C++ preprocessor directive or in this case more accurately a one-line macro definition, tells the compiler to replace any occurrence of the identifier (for example Rosco) with the text provided on that line. For example, whenever the compiler sees the word Rosco it will replace it with FALSE (which in turn is defined as 0). Add a new line for your robot. Declare it TRUE and the remaining robots FALSE. For example

#define BiPed TRUE

#define PaperBot FALSE

In a similar fashion add any sensors or actuators unique to your robot.

Next, open the command tab and scroll down to the command block you are implementing. The following example assumes you are modifying the MOVE command.



Append a #elif preprocessor conditional directive (before #endif). Within this new conditional block, add a function call to your unique command implementation.

#elif BiPed

walk\_BiPed(data);

Save and close the Arduino IDE. Add the ino file(s) containing the functions that are unique to your robot.

Reopen the Arduino IDE. The new ino file(s) will now appear as tab(s). Remove any setup() or loop() methods from your development software.

Verify (compile) your code and verify that there are no errors.

#### Unpacking/Formatting Command Arguments

As mentioned earlier, all command data is in the form of a byte array, formatted as an Arxterra command packet. Your unique commands may want data formatted as a type other than an 8-bit byte. In this example, four (4) bytes are converted into a 32-bit “unsigned long” data type and saved in variable ping\_interval.



#### Command Exception Codes

# Telemetry

## Testing Robot Commands

It is often desirable to test commands locally before testing remotely. This removes the potential problems introduced by adding smart device, internet cloud, and Arxterra control panel layers. To test locally you will need a terminal application program. The following instructions use the [CoolTerm](http://freeware.the-meiers.org/) program.

Mapping pins

ATmega2560-Arduino <http://arduino.cc/en/Hacking/PinMapping2560>

ATmega328-Arduino <http://arduino.cc/en/Hacking/PinMapping168>

Reminder(s)

Set Serial Monitor Baud Rate to match communication standard (usb = 57600, Bluetooth v2 = 9600)

To Do List

1. Pingers code to stop rover included but not tested/debugged.
2. Need to write code for pan/tilt offset angle.
3. Continue to clean up code.

Servo Notes

* The min pulse width (default is 544 microseconds) corresponds to the minimum angle (0-degree), shaft rotates fully left.
* The max pulse width, (defaults to 2400 microseconds) corresponds to The maximum angle (180-degree), shaft rotates fully right - rotation is left-to-right clockwise.

ATmega internal temperature sensor

#define ATMEGA\_TEMP 8

http://forum.arduino.cc/index.php/topic,38043.0.html

http://forum.arduino.cc/index.php/topic,26299.0.html

Problem is time required when you switch between voltage reference sources for ADC to stabilize. This would limit how often we check this internal sensor.

ArxRover needs to add cell phone temperature.

Saved in UserState File on Android Phone

public var cameraAdjustForMotion:Boolean = false;

public var cameraCanPan:Boolean = true;

public var cameraCanTilt:Boolean = true;

public var cameraConfigDefault:CameraConfig;

public var cameraConfigMotion:CameraConfig;

public var cameraIndex:int;

public var roverName:String;

Defaults set in the Control Panel's Control Options pop-up Window

Duty Cycle Steps 6 (4 -12)

Polling msec 500 (300 - 800)

Minimum Duty Cycle 100 (10 - 140) \*\*\* ArxRover \*\*\*

Top Duty Cycle 212 (180 - 255)

Motion State Display ON (Motion Text Display)

Control Tips Display ON (Tool Tips)

Range Sensor Display ON (Numeric and Graphical Icon of Ultrasonic Data)

While Loops

Waiting in a while loop is always dangerous you may want to consider adding a loop count exception or check for memory available. <http://playground.arduino.cc/code/AvailableMemory>

# Testing

## CoolTerm

Windows D - Show/Hide everything on local desktop

Left button

Settings

Accounts and Sync

Add Account

Options

Terminal - Line Mode

Local Echo

Misc - Auto ON/OFF

Window 1 - Connection - Send String

Window 2 - View Hex

Save STC

To send Hex Connection from menu bar and Send String and click Hex radio button.

## Test Cases

### Move (0x01) Forward Half Speed Test

**i = 0 1 2 3 4 5 6 7**

**N = 1 2 3 4 5**

**A5 05 01 01 80 01 80 A1 MOV forward half speed**

sendPacket

data[0] A5 = Command Packet ID

data[1] 05 = Packet Length N w/o parity byte

data[2] 01 = Move ID

data[3] 01 = Left Motor Forward

data[4] 80 = Half Speed (128/255)

data[5] 01 = Right Motor Forward

data[6] 80 = Half Speed (128/255)

data[7] A1 = Longitudinal Redundancy Check (LRC)

### Camera Home (0x04) Test

command = camera home w/ incorrect parity

**A5 01 04 00**

Command Packet ID A5

N = 1

Camera Home = 4

Parity 00 = should be A0

telemtry

**CA 03 0E 03 A0 64**

sendPacket

data[0] CA = Telemetry Packet ID

data[1] 04 = Packet Length N+1 sendWordPacket

data[2] 0E = Exception ID = id

for(uint8\_t i = 0; i < N=2; i++)

data[3] 03 = LRC Checksum Error = data[0] = high byte of word sent

data[4] A0 = Expected Checksum = byte[1] = low byte of word sent

data[5] 64 = Packet Checksum

N = 2 <= length argument

command = camera home w/ correct parity

**A5 01 04 A0**

telemetry

**CA 02 0D 04 C1**

data[0] CA = Telemetry Packet ID

data[1] 04 = Packet Length N+1 sendWordPacket

data[2] 0D = Command Echo

data[3] 04 = Camera Home

data[4] C1 = Packet Checksum

### EEPROM Write (0x07) Test

Write Command

**A5 14 07 00 00 10 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 00**

Start Byte A5

N = 20 or 0x14

EEPROM Write Command 07

EEPROM Address 0x0000 00 00

Number of Bytes 16 10

Data 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10

Parity Byte 00

# Appendix Arduino MEGA ADK Support

Arxterra code traces its origins to the Pathfinder rover which used an Arduino ADK and communicated over the Android Device Bridge (ADB), which is no longer supported by Google. Current Arxterra class robots use USB OTG, Bluetooth v2, and in the future Bluetooth LE for communications. Therefore for readability, legacy ADB code elements have been deleted. The last version to with support for the USB ADB is paperbot\_v6.

## How to Update Adb Library to Support Arduino version 1.0+

The Adb library has the following modifications made to bring it into compliance Arduino version 1.04 (Compiles without errors)

1. Adb.h, usb.cpp, and max3421e.cpp
2. Change the line: #include "wiring.h" to #include "Arduino.h"
3. Runing on ADK generates a "OSCOKIRQ failed to assert" error. See http://forum.arduino.cc/index.php?topic=68205.0

If using the ADB Library found in the arxterra Github then these changes have already been made.