Xaxxon Project – Oculus Robot Documentation – Part 3

Version 1.1

July 18, 2017

Research Co-op for Dr. Shahram Payandeh

Written by Lestley A. Gabo

Contents

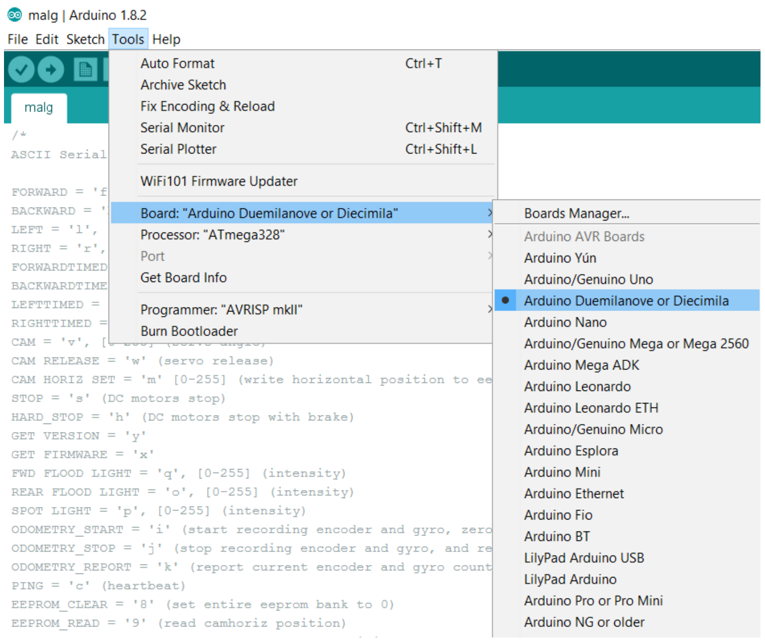
Connecting the MALG board to the Oculus robot

If it is not already installed, download and install Arduino IDE sketch. This is found in the Ubuntu Software Center.

The Xaxxon GitHub page (https://github.com/xaxxontech/malg) has a download link for a sketch or .ino file of the MALG board firmware and an I2C library that it requires. It is not necessary to download those. The firmware from the Xaxxon GitHub page is a malg.ino file that contains the code uploaded into the onboard MALG, the MALG inside the Oculus robot. While the I2C is a required library for the malg.ino because it won't run without I2C being imported.

We are using our own code or .ino file, provided by Jae, to control our servo motors. What is important from the Xaxxon GitHub page is the setup of the Arduino IDE sketch:

Set board type to 'duemilanove' Set processor type to 'ATmega328'



Board: "Arduino Duemilanove or Diecimila"
Processor: "ATmega328"
Programmer: "AVRISP mkII"
● Arduino Duemilanove or Diecimila

```
/*
ASCII Serial

FORWARD = 'f
BACKWARD = '
LEFT = 'l',
RIGHT = 'r',
FORWARDTIMED
BACKWARDTIME
LEFTTIMED =
RIGHTTIMED =
CAM = 'v', [
CAM RELEASE = 'w' (servo release)
CAM HORIZ SET = 'm' [0-255] (write horizontal position to ee
STOP = 's' (DC motors stop)
HARD_STOP = 'h' (DC motors stop with brake)
GET VERSION = 'y'
GET FIRMWARE = 'x'
FWD FLOOD LIGHT = 'q', [0-255] (intensity)
REAR FLOOD LIGHT = 'o', [0-255] (intensity)
SPOT LIGHT = 'p', [0-255] (intensity)
ODOMETRY_START = 'i' (start recording encoder and gyro, zero
ODOMETRY_STOP = 'j' (stop recording encoder and gyro, and re
ODOMETRY_REPORT = 'k' (report current encoder and gyro count
PING = 'c' (heartbeat)
EEPROM_CLEAR = '8' (set entire eeprom bank to 0)
EEPROM_READ = '9' (read camhoriz position)
```

Figure 1: set up board type 'Duemilanove'

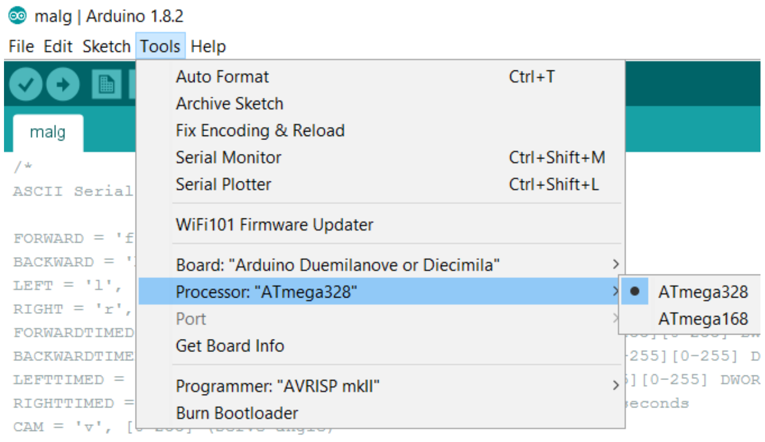


Figure 2: set up processor type 'ATmega328'

When the MALG board is plugged in to the Oculus robot, the Port/Serial Port will automatically show the new device. If you plug the board in and the port is open, you can just upload right away.

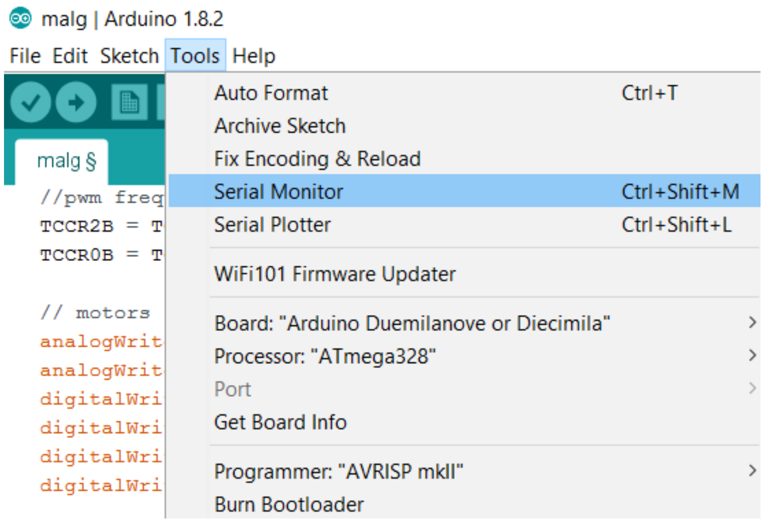


Figure 3: Serial Monitor

Most of the time when the MALG board is plugged in, the Serial Port it uses is /dev/ttyUSB2, but sometimes it changes to /dev/ttyUSB0, /dev/ttyUSB1, /dev/ttyUSB3, or /dev/ttyUSB4. To solve that issue, see section **Setting up persistent USB port path**.

Testing the MALG board using the Arduino IDE

The MALG board needs to be tested and the uploaded .ino code needs to be optimized. To upload new code into the board the steps below need to be followed because the Arduino IDE must be the one connected to the MALG board and not the node server.

If your node server is on, turn off the node server with CTRL+C in the terminal or close the terminal. The node server listens to the port if it is open, close, or busy so the IDE sketch cannot communicate with the board as it is busy.

If you have the pm2 set up, then you can turn off the node server this way:

Open the terminal and type in

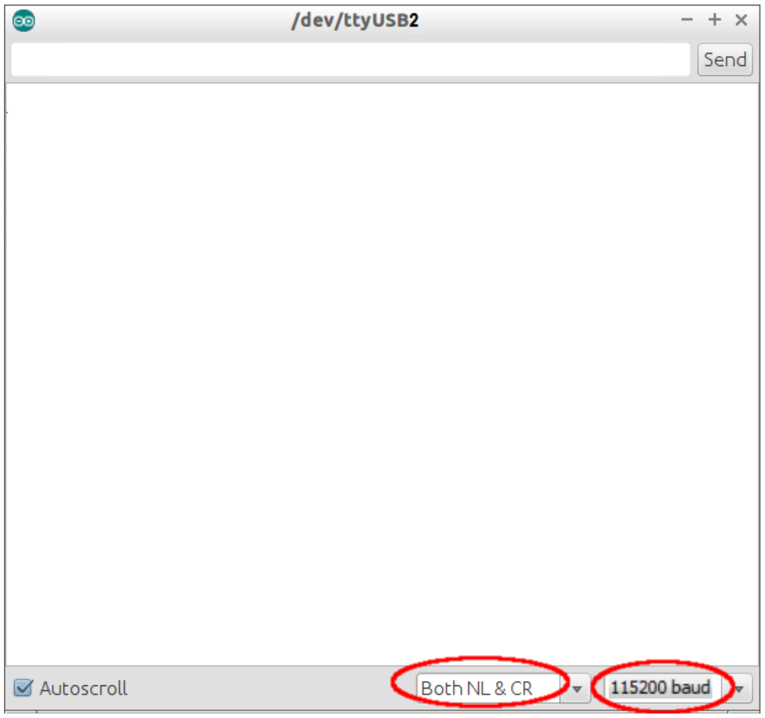
```
pm2 list
```

This lists the node servers that are on, choose to turn off the name of your server that communicates with the MALG board. In our case we stop our server with the line below because our server is named 'server'.

```
pm2 stop server
```

After uploading the new code into the board, you want to see the outputs from Serial.println("string") so open the serial monitor.

Make sure to replace on the bottom right of the serial monitor the **No line ending -> Carriage Return** and the baud rate to the one in your code's setup. In Figure 4 below the baud rate used is **115200 baud**.



Both NL & CR
115200 baud

Figure 4: change no line reading to CR and use proper baud rate

Installing NodeJS

There are various ways to communicate an Ubuntu OS with a MALG board. NodeJS is used in this project. A node server is created enabling communication between the MALG board and the web application. NodeJS can control the MALG board from the web page as if writing inputs into the IDE sketch Serial Monitor.

NodeJS also has a library management system called node package manager (npm) that extends its functionality besides being a server program.

NodeJS is being installed with Node Version Manager (nvm). The steps below are based on the tutorials from these two links:

http://www.hostingadvice.com/how-to/install-nodejs-ubuntu-14-04/#node-version-manager

https://itp.nyu.edu/physcomp/labs/labs-serial-communication/lab-serial-communication-with-node-js/

Open the terminal and type in

```
sudo apt-get install build-essential checkinstall

sudo apt-get install libssl-dev

sudo apt-get install curl

curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash
```

Check to see if nvm was installed with

```
nvm --version
```

Now install NodeJS (Note: instead of “nvm install 5.0”, “nvm install node” installs the latest node version)

```
nvm install node
```

Check the node version using

```
node --version
```

And then tell nvm to use that version of node (our latest version of NodeJS was 8.1.2)

```
nvm use 8.1.2

nvm alias default node
```

How to set up the node server

For reference, the tutorial link below was followed to set up our node server.

https://www.youtube.com/watch?v=G0BzzuXS8gI

Our node server is set up in **/home/oculus/push_to_git/xaxxonproject/malg-index2-ajax/**. To make a new node server follow the steps below.

Open the terminal and type in

```
cd push_to_git/xaxxonproject

mkdir “new_node_server”
```

Name the made directory to your own choosing without the quotes.

```
cd “new_node_server”

npm install ajax-request –save

npm install serialport

npm init -y

npm i -S express body-parser

touch jquery.js
```

Inside jquery.js copy paste the minified code from jQuery CDN (https://code.jquery.com/jquery-3.2.1.min.js)

```
touch server.js

touch index.html

touch scripts.js
```

The code for our server.js have artifacts of code from the tutorial above and most of them are unrelated to MALG communication. Also, instead of linking the source for the jquery.js into the index.html, the method used in the tutorial allows for offline use of jquery by copy pasting the jQuery into a local file.

The codes for server.js, index.html, and scripts.js are not included here. See section **How to use AJAX to control our MALG** for more information about the codes in these files.

How to use Serialport library

Serialport library is used to communicate with our MALG board. The Serialport library contains functions such as open, data, close, and error.

To import the Serialport library include in

```
var SerialPort = require('serialport');
```

Then create a pertinent variable such as 'myPort' that contains the port path of the MALG board, the baudRate, and a parser to read out the commands.

```
 var myPort = new SerialPort('/dev/arduino', {
          baudRate: 115200,
   // look for return and newline at the end of each data packet:
   parser: SerialPort.parsers.readline('\r\n')
});
```

The port path is "/dev/arduino" and not "/dev/ttyUSB2" because of section **Setting up persistent USB port path**. The baudRate is 115200 because that is the baudRate used in the onboard MALG.

After setting up 'myPort' we then use the Serialport events.

```
myPort.on('open', showPortOpen);
myPort.on('data', sendSerialData);
myPort.on('close', showPortClose);
myPort.on('error', showError);
```

If the port path is open then function showPortOpen is called.

```
function showPortOpen() {
          console.log('port open. Data rate: ' + myPort.options.baudRate);
          setTimeout(commandReady, 2000);
}
```

The setTimeout inside showPortOpen means it will wait 2000 milliseconds or 2 seconds and then calls the commandReady function. The MALG board needs time to turn itself on and 2 seconds allows enough time for the board to turn on.

```
function commandReady(){
          console.log('Ready to receive camera commands: up, down, left, or right.');
}
```

There are many command functions and they follow a pattern. For example, 'commandX()' below:

```
function commandX(){

        myPort.write("x \r");

}
```

The ‘write’ Serialport event is used. Write event is like a user typing into the Serial Monitor of an IDE sketch. The inputs “x \r” means the letter “x” is being sent, while the “ \r” sends a Carriage-return another term for return or enter. Without the Carriage-return “ \r” the command does nothing because like in a Serial Monitor, the enter key is not being pressed.

The Serialport only allows communication with the MALG board. See section **How to use AJAX to control our MALG** to see the pertinence of ‘commandX()’.

How to use AJAX to control our MALG

AJAX = Asynchronous JavaScript And XML. AJAX is used because it allows requesting and receiving of data from a node server. AJAX also allows sending data to a node server.

For reference, the tutorial link below was followed to set up our AJAX requests.

https://www.youtube.com/watch?v=G0BzzuXS8gI

Inside index.html

Inside the index.html there are buttons for each command with ids for their respective command.

```
<button id="get-commandX">commandX</button>
```

This index.html page is the view page when the node server turns on. It is at URL address

localhost:5000 or http://207.23.183.201:5000/

Inside scripts.js

The command ids are used in scripts.js where the ids are used to identify each command. When a button is pressed, AJAX will look for the id being pressed and sends a GET request to the node server in server.js with a specific URL tied to that id. Inside scripts.js we have

```
//GET/READ
        $('#get-commandX').on('click', function() {
                $.ajax({
                        url: '/commandX',
                        contentType: 'application/json',
                        success: function(response) {
                                var tbodyEL = $('tbody');

                                tbodyEL.html('');
                        }
                });
});
```

There is a lot of extra content after “url: ‘/commandX’,”. Since what is important is just sending a GET request to the node server, the code can be cut down to:

```
//GET/READ
        $('#get-commandX').on('click', function() {
                $.ajax({
                        url: '/commandX'
                });
});
```

In summary, the function above reads a ‘click’ from id ‘get-commandX’ and then sends that to server.js as a GET request for url ‘/commandX’.

Inside server.js

Inside our server.js we have

```
var PORT = process.env.PORT || 5000;
```

This sets up the port of the node server web application so the view page opens at localhost:5000.

Then there are the GET receivers that function after receiving a GET request

```
app.get('/commandX', function(req, res) {

        commandX();

        res.send('Successfully sent command x!');

});
```

From scripts.js our GET request is captured on ‘app.get’ above. When server receives a GET request ‘/commandX’ then it will run the function ‘commandX()’ and send back a console input of 'Successfully sent command x!’.

The function ‘commandX()’

```
function commandX(){

        myPort.write("x \r");

}
```

From section **How to use Serialport library**, the ‘write’ Serialport event is used. Write event is like a user typing into the Serial Monitor of an IDE sketch. The inputs “x \r” means the letter “x” is being sent, while the “ \r” sends a Carriage-return another term for return or enter. Without the Carriage-return “ \r” the command does nothing because like in a Serial Monitor, the enter key is not being pressed.

Installing production process manger for NodeJS (pm2)

Production process manager, pm2, allows the node server to start automatically on boot. There would be no need to keep opening the terminal and typing in “cd ‘directory’” and “node ‘server name’” because the process manager starts the node server automatically.

Open the terminal and type in

```
npm install pm2 --g
```

```
pm2 startup
```

This creates a command that needs to be copied and pasted, below is example of what it creates.

```
sudo env PATH=$PATH:/home/oculus/.nvm/versions/node/v8.1.2/bin
/home/oculus/.nvm/versions/node/v8.1.2/lib/node_modules/pm2/bin/pm2 startup
ubuntu14 -u oculus --hp /home/oculus
```

To start the node server (Note: This is the directory for ours and our node server’s name is ‘server’.)

```
cd /home/oculus/push_to_git/xaxxonproject/malg-index2-ajax
```

```
pm2 start server.js
```

To save the process

```
pm2 save
```

To check the node app being run in the process manager (pm2)

```
pm2 list
```

```
pm2 info server
```

To check the console log of the node server

```
pm2 logs server
```

If ‘pm2 list’ shows your node server as ‘status: online’ then after a ‘sudo reboot’ or ‘sudo halt’ the node server will be turned on automatically.

When the file server.js is edited, then the pm2 must be restarted

```
pm2 restart server
```

If a new .ino code needs to be uploaded into the MALG board, the node server needs to be stopped because it is listening in on the MALG board so the MALG board port is busy.

```
pm2 stop server
```

Then start it again with

```
pm2 start server
```

Setting up persistent USB port path

Our goal is to set up a unique port path for our MALG board. This is done by creating a new dev rule that links whatever port path the MALG board is currently connected to with the port path '/dev/arduino'.

Figure out the port the MALG Is using **from the Arduino IDE sketch**. In this example, the MALG board is in port /dev/ttyUSB2.

Tools > Serial Port > /dev/ttyUSB2

Open the terminal and type in

```
udevadm info --name=/dev/ttyUSB2 --attribute-walk
```

The most important ATTRS is the serial, so specify the serial with

```
udevadm info -a -n /dev/ttyUSB0 | grep '{serial}'
```

The list below contain the pertinent information:

```
SUBSYSTEM=="tty"

ATTRS{idVendor}=="10c4"

ATTRS{idProduct}=="ea60"

ATTRS{serial}=="008C8662"
```

Create a file /etc/udev/rules.d/99-usb-serial.rules

```
cd /etc/udev/rules.d

sudo touch 99-usb-serial.rules

sudo nano /etc/udev/rules.d/99-usb-serial.rules
```

Then inside the 99-usb-serial.rules type in

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea60",
ATTRS{serial}=="008C8662", SYMLINK+="arduino"
```

Turn on the rules

```
sudo udevadm trigger
```

Verify the ardiuno link

```
ls -l /dev/arduino
```

The verification should look like this

```
lrwxrwxrwx 1 root root 7 Jul  7 17:10 arduino -> ttyUSB2
```

It is then confirmed that port path '/dev/ttyUSB2' was linked to '/dev/arduino'. Therefore, whichever USB port our MALG is plugged into, its port path will always be '/dev/arduino'.

References

The Xaxxon GitHub page - https://github.com/xaxxontech/malg

Figure 4 - http://www.makeloft.org/2016/04/bluetooth-controlled-redbot-construction.html

Installing NodeJS - http://www.hostingadvice.com/how-to/install-nodejs-ubuntu-14-04/#node-version-manager

How to use Serialport - https://itp.nyu.edu/physcomp/labs/labs-serial-communication/lab-serial-communication-with-node-js/

How to use AJAX - https://www.youtube.com/watch?v=G0BzzuXS8gI

jQuery CDN code - https://code.jquery.com/jquery-3.2.1.min.js