# Robotics platform

## Architecture 3, Test 2/3:

* This design uses the databases (masterControl and status) as the primary data exchange medium on the server side. I think if possible, the use of zMQ from the human interface to the controller may reduce/eliminate lag. Since I have made the decision to hard code play modes (masterControl commands), the need for a database to archive all commands and used to ‘’select’’ the active command would not be necessary.
* The start/stop commands are working.
* Is there a need for user controlled commands (other than game mode). Such as : robot 1 slow down. Or is it okay to let the server controller micro-manage specific actions (caused by hall effect switches and rfid tags – through the “special1” and “special2” fields).
* Only basic functionality has been implemented to demonstrate the viability of this architecture. So far tests have been positive. Delays are minimal (even with heavy db use on server side).

**Data format from server to robot:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | drive | aux | Special1 | Special2 | report | request |

* **ID**: This needs to be larger than “0” (zero) for the controller scrip to read the masterControl database. By default, use “1”.
* **Drive**: If drive is “1” and request is “4”, the robot is enabled to drive normally.
  + If drive is “0” and request is “4”, the robot is stopped.
* **Aux**: This selects if the auxiliary motors are enables (“1”) or disabled (“0”). Not implemented.
* **Special1, special2**: special commands where RFID tag/hall effect switch responses can be sent back to the robot. Depending on the play mode selected by the control user, these can mean different things. The controller has to implement special1&special2 capabilities in the future.
* **Report**: Refers to the robot number (1..16). Use “60” for all robots.
  + This selects either a specific robot to which this command applies or selects to send the command to all robots.
* **Request**: Details what type of request the command line has.
  + Request = “1” : return command values to null (default driving mode)
  + Request = “2”: Send robot status to the server.
  + Request = “3”: special 1 is used to be a number between “0” and “1”. This will be multiplied with the drive speed limit to slow a robot by a proportional factor of that value.
  + Request = “4”: will either enable the robot to drive normally (drive = “1”) or stop the robot (drive = “0”)
  + FUTURE DEVELOPMENT EXPECTED.

Data format from robot to server.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| name | cdate | ctime | enmotors | pwml | pwmr | pwma1 | pwma2 | report |

* **name**: Name of the robot such as (“robot1”, “robot2”,…, “robot16”). No spaces!!!
* **cdate**: The date
* **ctime**: The time at which the data was sent from the robot.
* **enmotors**: If “1”, motors are enabled. If “0” motors are disables.
* **pwml**: Pulse width modulation left. Shows values from 0 to 100 for pwm of left motor.
* **pwmr**: Pulse width modulation right. Shows values from 0 to 100 for pwm of right motor.
* **pwma1**: Pulse width modulation auxiliary 1. Shows values from 0 to 100 for pwm of aux 1 motor.
* **pwma2**: Pulse width modulation auxiliary 2. Shows values from 0 to 100 for pwm of aux 2 motor.
* **report**: no specific function at the moment.

CAUTION: Programmers should take note that: the robot script MUST have the sleep delays in the main loop. This is to reduce CPU load and decrease heat on the RPI. Removing the delays may cause overheating and/or catastrophic failure.

CAUTION: All values used are text based. They are surrounded by quotes ie: “1”. To use as a variable of type int, float etc you need to convert from string to int,float,etc.

### GUI notes and how to: (coded using Tkinter for python)

The graphical user interface (starting arch3\_test3) has real time robot status shown in the tables. To the left, you can enter the number of the robot (1..16) for which you want to commit changes(use “60” or leave blank for all robots). To the right, you can enter a number (float) between 0 and 1. This is a factor of speed which can be committed to slow down a robot (or all). Stop, Start can be used with robot selection. Slow down can use robot selection and factor of speed.

-Future improvements: - Have a stop/start button instead of using RadioButtons with a commit button.

- Sorted list (the list shows in order of who was last updated right now).

**How to:** To open the GUI, open a command window, navigate to the directory where the GUI .py file is located then open it using the command: *sudo python filenameGUI.py*

### System description + how to get started:

1. To run the architecture, the server has to have mysql and zmq installed.
2. The server must have a database called “Controller” and two tables. One called “status” and the other called “masterControl”. Names are case sensitive. See what column names to use above. Be careful, they are case sensitive.
3. The server must run the 2 control scripts (controller and GUI) as well as all of the server\_client scripts (one per robot). There is space for up to 16 robots, this can be altered if more are needed (resource permitting). System tested with 2 robot at the moment (8/3/2016). Remember than increasing sleep time within the server files may increase available resources to help add more robots.
4. You need the IP address of each robot as well as the IP address of the server.
5. Update the IP address of the server (current: 131.202.14.109) to your own in the python scripts. It is present in the PS3\_client files (one per robot). Each file has reference to a robot number (topicfilter/generaltopicfilter/targettopicfilter/aname). Use “robot1” to “robot16”. Each robot has to have distinct names. Make sure you rename the robots accordingly when creating the robot files. Update robot names in the corresponding server\_client\_robotX.py files.
6. In each server\_client\_robotX.py files, you need to insert the IP address of the robot. EX: if server\_client\_robot1.py, then the IP address replacing the current IP (131.202.14.140) will the be IP of your robot 1. Do this for all robots.
7. NOTE A: in server\_client : this port number needs to be different for each robot used. Update here and on the PS3\_client file on the targeted robot. I suggest using 5551(robot1),5552(robot2)…5516(robot16).
8. There is no specific order to open the files. Just make sure all of them are opened before starting to send commands.

TIP: To make yourself comfortable with the system. Set up 2 robots and observe how it works. There are two rpi with the complete install on them for the robots and there is a server also ready. Files are found in the “Markus” folder. Just connect the RPI, get their IP’s. Change the IP’s in the files of the Architecture 4 Test 1 and try the whole system. Open each file in a command window through putty. Open the GUI by connecting the server to a monitor and open the GUI file from a command window there.

**TIP2: Look at the diagrams of Architecture 3 Test 2. That is how the current design works. This shows from and to for all messages and how all the files fit together. This is the best way to understand the system**

# Architecture 4

This architecture focusses on creating homogeneous files that do not require modification regardless of the name/IP of each robot. The IP address of the server still has to be known.

### Login

The new design implements basic **login** between the robots and the server.

1. Robot will try and access the remote database. There is no bypass in the 8/4/2016 upload.
2. The robot will write “25” into the report field of the status table. This indicates to the server that the robot wants to sign-in.
3. After a successful insert into the database, the robot exits the log-in loop and looks for a PS3 controller. Then the main loop is executed. This is when the robot starts sending status updates to the server.
4. The server, in the meanwhile, was waiting for a field in the same database with which the report field is “25” and deleted all other rows (with name field = robotName argument).
5. Once the server has verified the information received it exits the log-in loop and proceeds with updating the robot status row with the information received from the robot.
6. The server will override the report field value of the robot status as soon as it exits the log-in loop. This is to prevent the server(once an automated script opens the client processes) to open multiple processes for a single robot. This is there as a fail-safe.

* Positive from this approach is that with MySQL, we know the info from the robot has been written and do not need to wait for a response from the server scripts before going further

-Note: fail-safe protection may be redundant. It is meant to offer the highest reliability possible during play and set-up.

### Running scripts is now slightly different.

* The PS3\_client file does not need to be modified: all robots will use the same exact file.
* The server\_client file does not need to be modified: all processe required to communicate with the active robots will use the same exact file.

This is achieved using system arguments as such:

PS3\_client (write in command line):

*~/sudo python Arch4\_PS3client.py robot2 131.202.14.109*

Server\_client (write in command line):

*~/Sudo python Arch4\_serverclient.py robot2*

Notice that robot2 is the first system argument and that the PS3client requires the IP address of the server! This can be automated through a boot loader and using a local server to fetch the required information. This will be done at a later time.