This is an implementation guide for connections between server (Raspi) and client (Android app).

## Handshake

### Client

### Server

## Initial Setup

### Client

### Server

### Hardware

## Throttle/Brake/Neutral

### Client

### Server

### Hardware

## Steering Left/Right/Neutral

### Client

### Server

### Hardware

## Left Turn Light (modify description for right/neutral also)

### Client

The ImageView will have an initial state, false because the left turn light should be off. When the user presses the image icon (ImageView) the client (Android app) will call a local function “setTurnLightState(direction=left, state=true/false)” which will change the initial state to the new one, change the image to ImageView and will contact server “set\_turn\_light\_state(direction=left, state=true/false)”.

*TODO find another implementation because this will have issues due to concurrency. Imagine the scenario where the user sets the light state to true and the thread to the client reads the previous state which will be false.*

*POSSIBLE SOLUTION 1:* A Thread (feedback\_info\_thread) will run continuously and will contact server “get\_turn\_light\_state(direction=left)” to get the current state from the server and call the “setTurnLightState(direction=left, state=true/false)”.

*POSSIBLE SOLUTION 2*: A server(NanoHTTP) will run at the client (Android app) which will receive a request from the server (Raspi) about the left turn light state whenever it changes. Then the client should call the local function “setTurnLightState(direction=left, state=true/false)” according to the request data it receives.

*POSSIBLE SOLUTION 3*: If method “set\_turn\_light\_state(direction=left, state=true/false” in server always returns a state from the hardware controllers “ElectricController.getTurnLightState(direction=left)” then the client will always be up-to-date with state (true/false) so there is no need to implement *possible solution 1* and *2*.

### Server

* 1. The server will be listening “set\_turn\_light\_state(direction=left, state=true/false)” for events. When an event arrives it will call the method on the hardware controllers “ElectricController.setTurnLightState(direction=left, state=true/false)”. Before exiting this method, it will get the state for the left turn light“ElectricController.directionMap[“left”]” and will return the result to the client (Android app) as an HTTPResponse to his HTTPRequest to turn on the left turn light “set\_turn\_light\_state(direction=left, state=true/false)”.
  2. This way I ensure that the state (true/false) has reached the hardware controllers and I get the state (true/false) from hardware controllers too. If, for any case, the left turn light will not turn on and the state has successfully reached the hardware controller it would be a hardware issue (controller software or the real hardware).

### Hardware

The “ElectricController” singleton will have a method “ElectricController.setTurnLightState(direction=left, state=true/false)” which receives commands from server only.

This class will have also a map “directionMap” ([“left”]→true/false, [“right”]→true/false, [“neutral”]→true/false) with initial values to “false”.

The method “ElectricController.setTurnLightState(direction=left, state=true/false)” will update (set every key to “false” value and then set the new value to the key from the method’s parameter, ex [“direction\_param\_value”]→state\_param\_value) the “directionMap” accordingly and then will apply the changes to the shift register hardware “ synchronized controlShiftRegister()”.

The “synchronized controlShiftRegister()” will get every value from the “directionMap” and update the led values accordingly. Pay **attention** to the different kind of boolean values, the boolean value which indicates if the user wants the left turn light to be on, and the local (to the method) boolean variable which controls the left turn light to change between on/off when the user wants to actually turn the left turn light on. To remember how the shift register works see my example at Tinkercad.