RF Remote Control for Models

Raspberry Pi - Android

by Bernd Hinze 2019

Release 1.1

2019-04-30

Content

Preamble

After retiring I was looking for a new personal challenge. Based on my experiences of the 80th and the current age of my grandchild I focussed on a remote control with an Raspberry Pi. Because I want to get some knowledge of Android application development the transmitter should be an application on the handy that controls the model.
Most of the text are translated with the usage of a web translator [Deepl].

1 Introduction

This documentation describes a remote control system. The receiver is a Raspberry Pi computer and a PWM module. It receives data and computes impulses for servos, speed controller and switches. The transmitter is an Android application, which can be adapted based on own requirements. The communication channel is based on a standard 2,4 GHz WiFi. The receiver software was developed with Python and the Android application of the transmitter with the Android mode of Processing [Processing]. Rebuilding, changing and using of this system force increasing the knowledge of two different computer languages and basic hardware knowledge especially for beginners. This baseline could be also an approach for a simple self moving robot (boat or car), caused by the sufficient resources for additional functions on the Raspberry Pi.

2 Requirements

At the beginning the project definition has to be created. The question: “What shall the device or Android application provide. What kind of tools shall be used to achieve this target. These requirements are normally divided in non-functional and functional requirements. The decision how something should be designed shall normally not be requested. Otherwise there are some constrains caused by the state of technology, well proved principals or the availability of hardware.

2.1 Overview

Figure 1 illustrates the overview of the system. The system consist of a Raspberry Pi as a receiver and a 12-Bit PWM module that is connected via the I2C interface. The PWM module drives up to 16 actuators like Servos, LEDs, relays and H-bridges. The transmitter is an application that runs either on a handy or during the test phase on a PC. Both components communicates via cyclic UDP-telegrams and must have access to the same local network via WiFi.

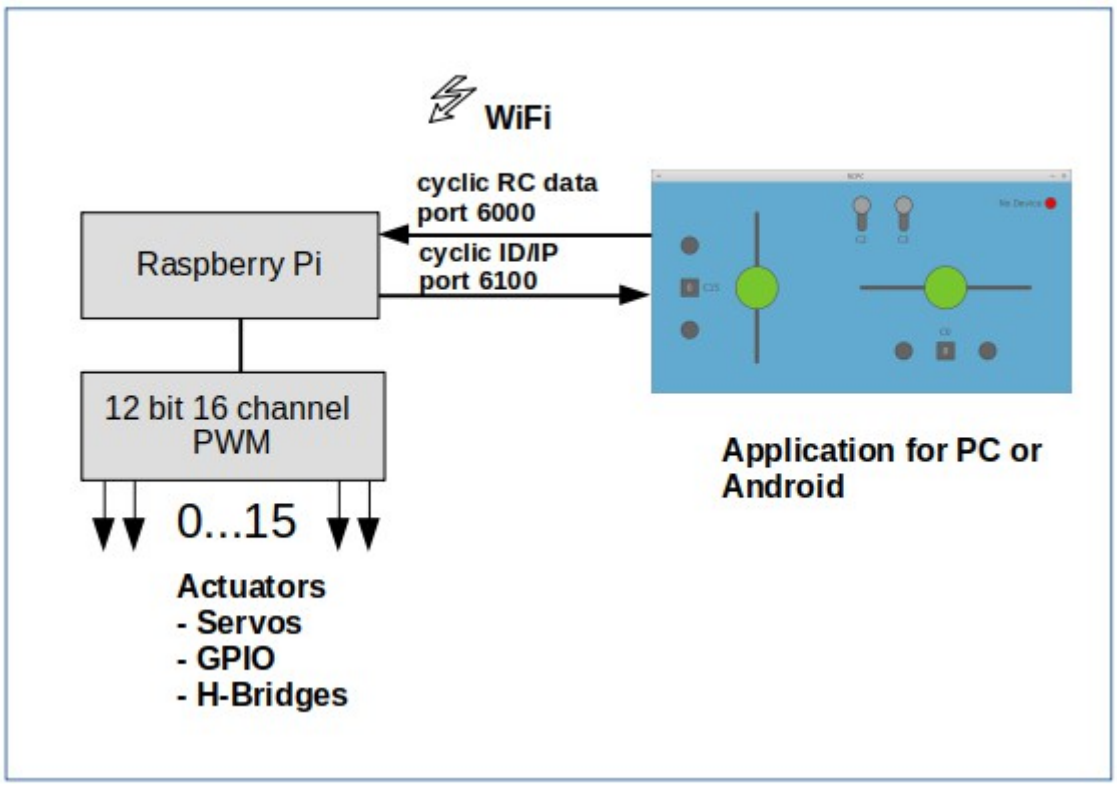


Figure 1: System Overview

2.2 Functional Requirements

Functional requirements provide a very good overview about the functions of the system. These requirements will be traced in larger projects from the requirement definition up to the validation of the system. That will be made normally by special labels. Caused by the limited functional amount and the usage in the hobby area without commercial goals, I do not use these method here.

2.2.1 Receiver

Following requirements shall be fulfilled:

- The application shall automatically start after the receiver has been switched on.
- The communication between the receiver and the transmitter shall use UDP-telegrams in a local WiFi network.
- The receiver shall extract and transform the received data for the interface of the PWM device driver.
- The receiver shall cyclically broadcast it's own IP in the local network.
- If the receiver does not receive any telegrams from the transmitter a special function, that can be adapted for own purposes, shall be carried out after a timeout.
  - Default action: Calling a fail safe function of the PWM controller following by a shutdown

of the Raspberry Pi.

- The receiver shall carry out a shutdown when receiving a special header number of the message.
- The PWM controller shall support following actuators
  - Servos with configurable duration of the impulses for lower and upper limitation.
  - Speed controller
  - Normal digital outputs with two states (ON/OFF)
  - H-bridge with the function of a speed controller
- The PWM controller shall have a fail safe function that forces the default position of the state of all actuators.
- The cycle rate shall be about 50 Hz (20 ms), but shall be configurable.
- The trim function for the servos shall shift the impulse look up table within preset limitations, without reducing the complete movement angle of servos.
- Max. 16 remote control channels shall be available.
- Each channel shall provide an inverted function witch shall be configurable.
- The timing behaviour of each channel shall be adjustable
- The fail save impulse value of each channel shall be adjustable

There may be further requirements witch shall be extended in a later release.

2.2.2 Transmitter

The transmitter shall full fill following requirements:

- The application shall support the adaptation of position and amount of following control levers:
  - Switches (default position - ‘OFF’),
  - Button for the the remote shutdown of the receiver application (default position - ON)
  - Control Lever with configurable default position and backward movement, if the lever is not touched.
  - Trim buttons
  - Trim value indication
  - State indication of the communication between transmitter and receiver
- The graphical elements shall be automatically resized by the height and width of the used screen.
- The application shall run either on a PC or an Android device. The differences of the code shall be as small as possible.
- Multi touch support shall be used for the Android application.

2.3 Interfaces

2.3.1 WiFi Interface

A simple cyclic UDP telegram is used for communication between transmitter and receiver. The configuration of the local network is not described here, as there are sufficient tutorials in the network. Sender and receiver must have a valid IP address in the local network.
The receiver listens on port 6000 for all UDP telegrams.
The assigned IP address of the receiver is determined and sent to the transmitter in a cyclic telegram - once per second. After the receiver IP is known by the transmitter, it sends control telegrams with a cycle time of approx. 20 ms. The structure of the telegram takes into account both the data definition of the 'MiniSSC' protocol [miniSSC] and the requirement for trimming and remote shutdown. In addition, own special commands can be implemented. Encoded ASCII characters are used for transmission.
The length of the telegram is dynamic and depends on the number of actuators in the transmitter. Each channel information consists of three bytes in the specified decimal value range.

Identifier	Channel Number	Set Value
0..255	0..15	0..254

Two values are implemented for the identifier:

255 : Transmission of the control value of a channel

127: Transmission of the trim value of a channel

100: The receiver shall carry out a shutdown

Up to 16 channel information can be transmitted in one UDP telegram.
The transmission of the trim data or of buttons and switches are event-driven. As soon as the value in the transmitter has been changed a telegram is transmitted. Before the UDP telegram is compiled, each byte is converted into two ASCII characters according to the following regulation:

H-Byte = CodeTabelle [Byte / 16]
L- Byte = CodeTabelle [Byte % 16]

Hex	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
String	’0’	’1’	’2’	’3’	’4’	’5’	’6’	’7’	’8’	’9’	’:’	’;’	‘<’	‘=’	‘>’	’?’
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Table 1: Code Table

Example:
Data to be transmitted (decimal):

255	0	127	255	1	120	255	2	30

As a result the transmitter sends following ASCII signs:
‘??007???0178??021’

2.4 Non functional Requirements

The following non-functional requirements shall be met:

- Development of the Android application should be done with the 'Processing' development environment. This allows the application to be developed and tested on the PC.
- Development of the receiver software with Python.
- The SW should have an object-oriented design. This is particularly useful for graphical elements that are to be used multiples.

3 Description

3.1 Hardware

3.1.1 Prototype

The hardware of the receiver consists of the following components:

- Raspberry Pi Zero W
- L298N H-Bridge or commercial speed controller
- 16-Channel, 12-bit PWM Controller Board with PCA9685
- Standard servo with JR-system or compatible connector
- Akku 7,2 V for the motor unit
- Optional: Step-Down Converter to provide 5 V for the Raspberry Pi
  or so called ‘BEC’ unit with 5 V output.

Figure 2 shows the block diagram and Figure 3 the experimental setup during development.
Different power supply concepts can be used.
As an alternative to figure 2, the drive motor can also be supplied with a higher voltage from a voltage source. In model making 7.2 V and 12 V are common.

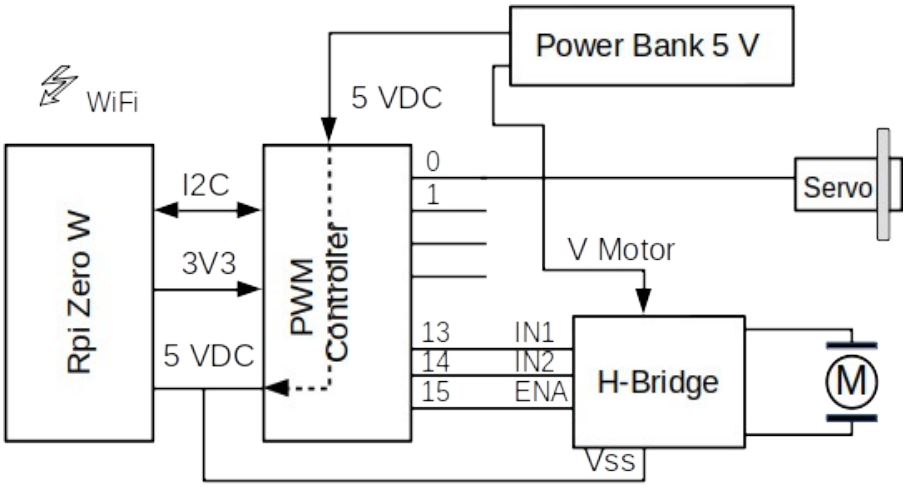


Figure 2: Block diagram of an exemplary configuration

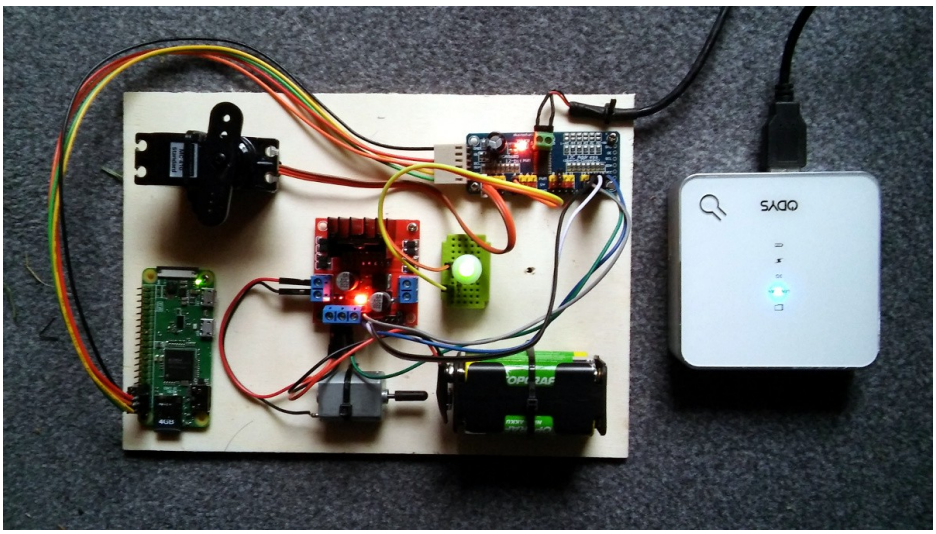


Figure 3: Prototype

Any Android compatible device can be used as a transmitter. USB debugging must be enabled on this device. Since this is not uniformly regulated, you have to look for the corresponding instructions on the internet or on the manufacturer's site of your device.
This is the solely way to transfer the application directly from the processing development environment to the mobile phone.

3.1.2 Integration in a Model

After an extensive search for an inexpensive beginner model, which also arouses some fun and interest in children with appropriate performance, I found a beginner model of 'Tamiya'. For less than 100 € you get a chassis platform with servo, drive motor 540 and a simple speed controller. The integrated speed controllers additionally provide a voltage of 6 V for the receiver. Since the Raspberry Pi requires a voltage of 5 V, an additional voltage reducer is required. Other than used in the prototype, I ordered from 'Adafruit' [AdaPWMBon] the new 12 Bit 16 Channel PWM Bonnet, which you can plug directly on the Raspberry Pi. The result is an extremely compact 16 channel

receiver. The block diagram is shown in Figure 4. Figure 6 shows the Raspberry Pi with 16 channel PWM Bonnet . Figure 7 illustrates the repair of an play car with an old 27 Mhz RF-system. Please be aware that the power supply of the Raspberry Pi rooted by the PWM-controller requires an additional shunt. Figure 5 pictures the shunt for a PWM-bonnet by ‘Adafruit’. The capacity ‘C2’ shall be at least 500 uF.

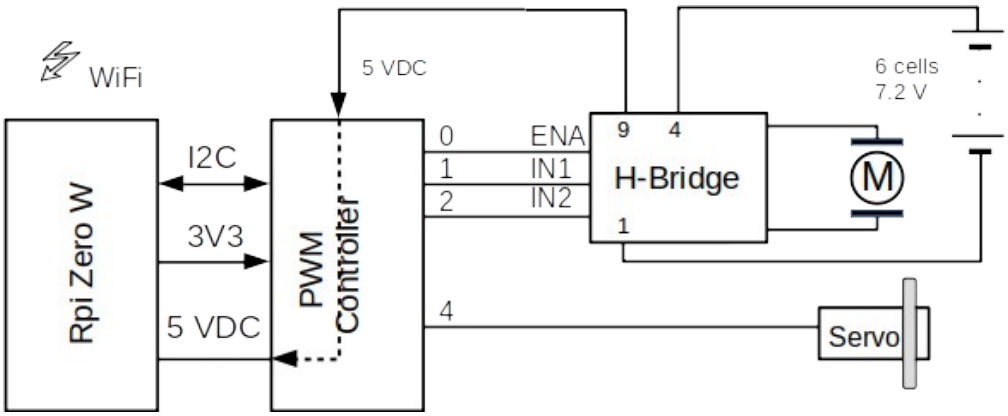


Figure 4: Block Diagram of used Components

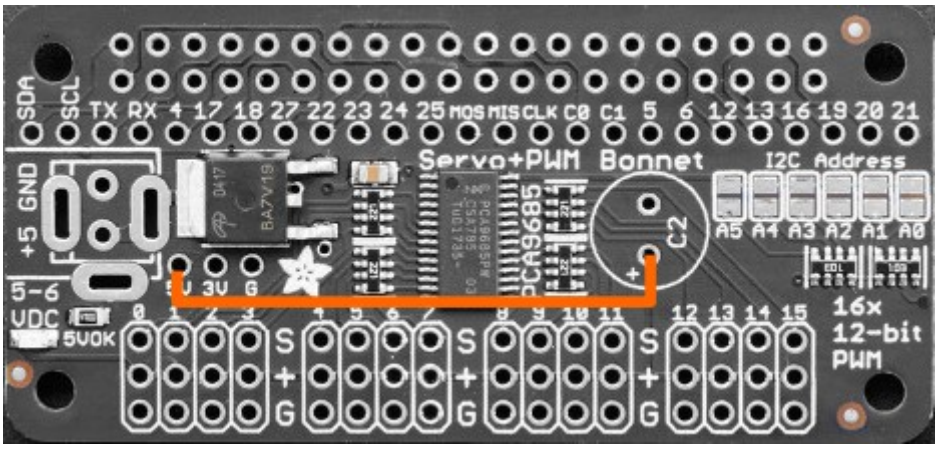


Figure 5: Shunt for Raspberry power supply by PWM module

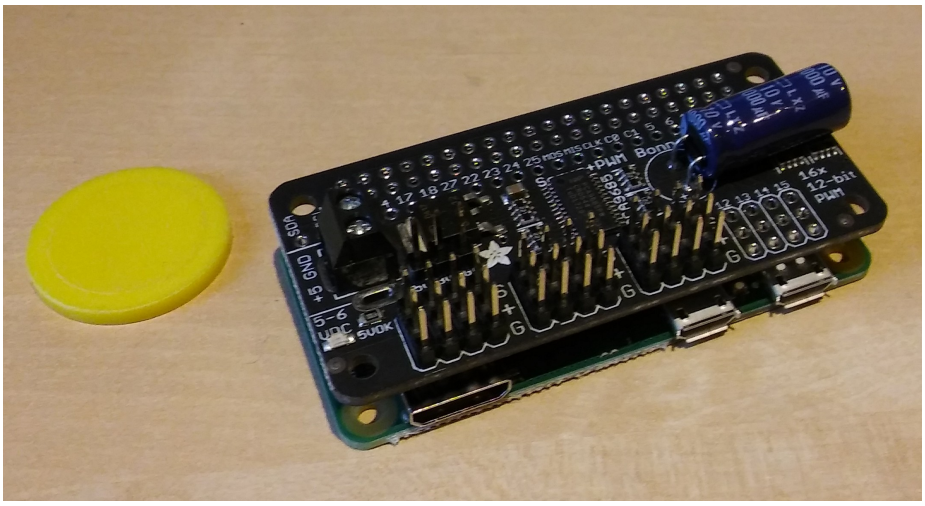


Figure 6: Raspberry with PWM Module

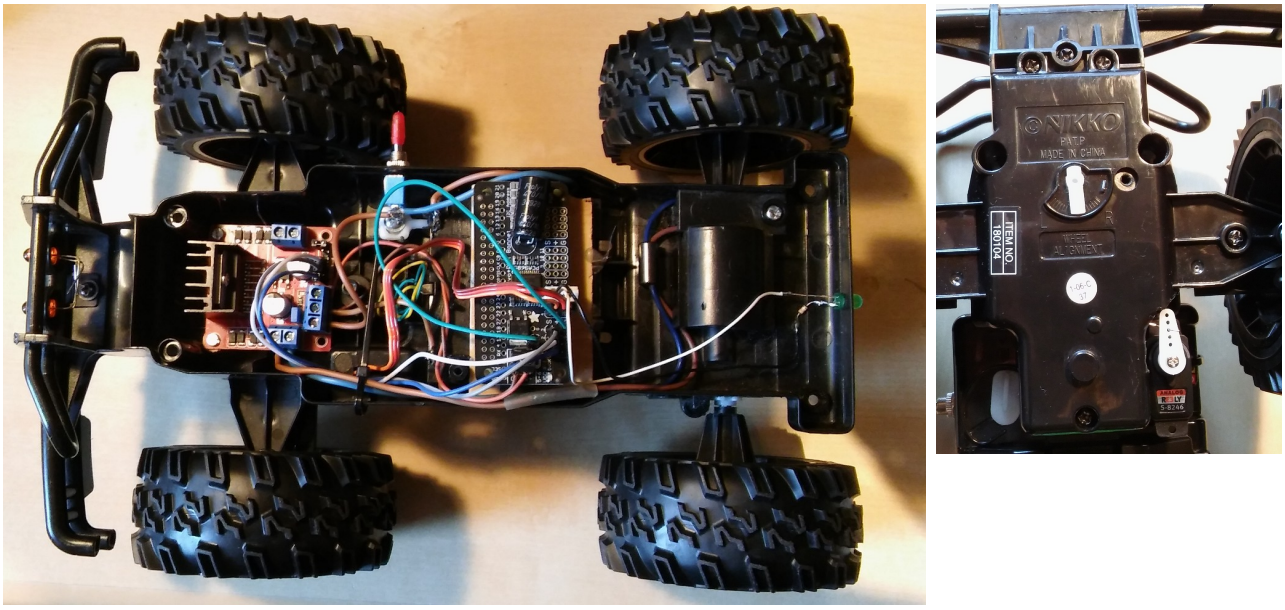


Figure 7: Old 27 Mhz Car, repaired with our system, Mini Servo added

3.2 Software

3.2.1 Sender (PC, Android)

This application was developed with Processing [Processing]. This Java development environment developed at first by MIT has an Android mode. This mode make it possible to test the essential components of the SW on the PC and then load them onto the Android device. If no special functions or libraries are used that are only available in Android mode, the application can be run on an Android platform without any changes. In our special case this was not 100% possible. The user interface of the Android application requires the usage of the 'Multi touch' functions. Therefore two processing sources are available, but they differ only insignificantly.

3.2.1.1 Installation of Processing

Processing is available for download for all common platforms (Windows, LINUX and IOS). There are two instructions which describe the installation and usage [ProForAndr] and [ProForBeg]. It is important to add the Android mode and install the UDP Library [UDPLib]. Libraries can be installed via the menu item "Tools -> Add Tools/".

3.2.1.2 Application

I didn't know Processing before and was a complete novice. But with the excellent web page [Processing] I managed to build up the necessary knowledge very fast. It was important that all graphical elements were developed object-oriented, so that a very uncomplicated change of the SW for own requirements is possible. All dimensions and positions important for the visualisation

were calculated with the system variables **`height`** and **`width`**, which serve as scaling factors. Thus, the elements automatically adapt to different screen sizes.

Following classes and functions have been implemented:

`String int2str(int inp):` Converts an integer value into two encoded strings.

`boolean overCircle(int x, int y, int radius):` Determines whether the mouse pointer or finger is over the area given by the parameters.

`Class Indicator`: Element that indicates the state of the communication

`Class Lever:` Control lever vertical

`Class LeverT:` Inheritance from class 'Lever', control lever horizontal

`Class Trim:` Trim button with indication

`Class Switch:` On/Off switch

`Class SwitchApp:` Inheritance from class 'Switch' with different labels for remote receiver shutdown

`Class CtlBu2:` Two control Button to steer very easy models without a servo

3.2.1.2.1 Class Indicator

Draws an indicator to indicate the status of communication between the receiver and transmitter. If cyclic telegrams are received, the receiver ID is also displayed.

3.2.1.2.2 Class Lever

The vertical control lever is implemented in this class. The function of the horizontal control lever is implemented in a derived class **`'TLever'`**.

Constructor:

```
Lever(int cx, int clim_pos_low, int clim_pos_high, int cdefault_Pos,int channel)
Parameter:
      cx: Distance from the left boarder
      clim_pos_low: Distance of the guide slot (top) to the upper boarder
      clim_pos_high: Distance of the guide slot (button) to the upper boarder
      cdefault_Pos: Default position of the knob in (0% .. 100%)
      channel: Channel of the PWM module
```

Furthermore additional variable will be used which are illustrated in figure 8.

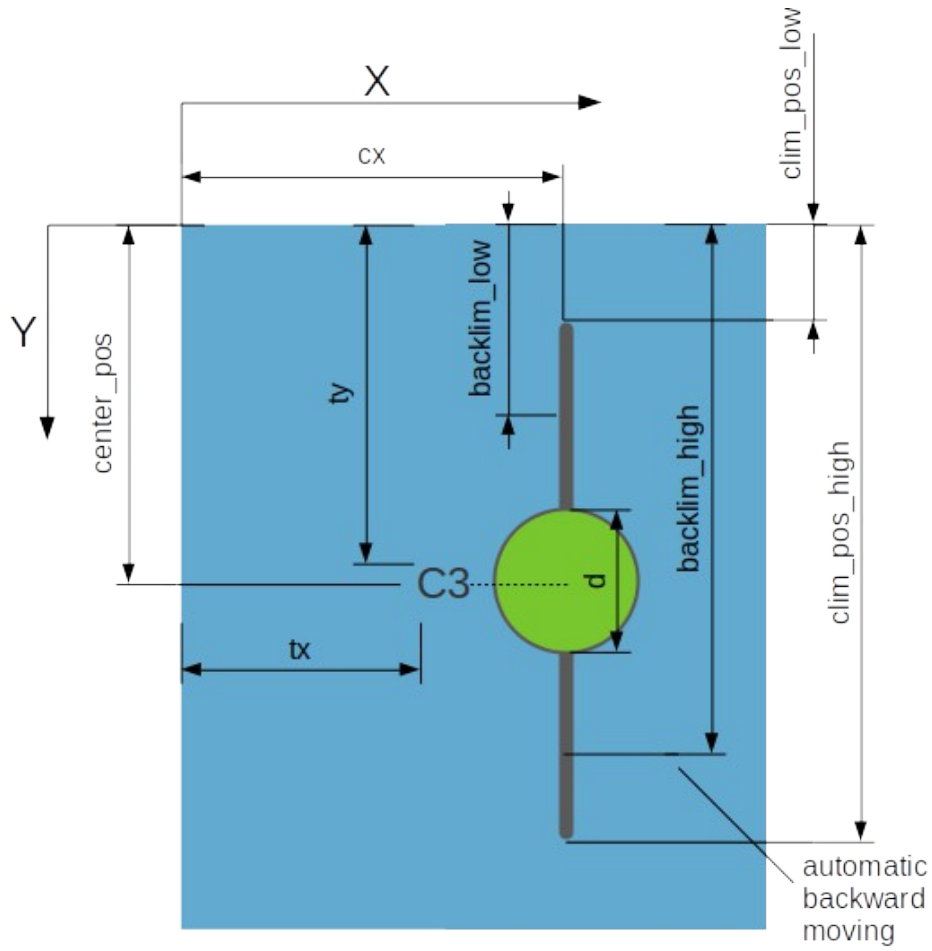


Figure 8: Control Lever Geometry

The class provides following methods:

`void display():`

- Drawing the guide line
- Display of the used channel
- Determining the current position of the mouse or finger (**`overCircle(x, y, radius`**)
- Determine the new position if there is contact, otherwise return the control lever to the home position.

`void LeverHandle(int X_pos, int Y_pos):`

- Drawing of the control lever on position X-pos, Y_pos

In addition to the graphical functions, there are two methods that calculate and provide the values to be transmitted. A value in the range 0..254 must be calculated from the position of the control lever. The value 127 must be returned for the home position. For this purpose the method **`createValMap()`** is called when the object is created by the constructor.

`void createValMap(int min, int max)`

- This function generates a table which maps the value range of the control lever (y-coordinates) to the permissible value range of the telegram (0..254).

String getVal(int setVal)

- This method returns the entire telegram to be sent. **SetVal** serves as an index to determine the output value to be transmitted from the **ValMap** table.

3.2.1.2.3 Class TLever

This class inherits all properties and methods from the **Lever** class. It represents the control lever horizontally. Only the constructor and the **draw()** method are overwritten to rotate the coordinate system.

3.2.1.2.4 Class Trim

This class is used to draw the trim elements and to generate the trim telegram.
Constructor:

Trim (int cX, int cY, String orientation, int channel)
Parameter:
cX : Centre position, distance from the left boarder
cY : Centre position, distance from the upper boarder
orientation: "L" | "P" ("L" = horizontal, "P" = vertical)
channel: channel of the PWM module, which should be trimmed.

Each time the mouse button is pressed or one of the trim buttons is touched, the preset value is increased or decreased. The value range for the value to be transmitted is 0..50. The mapped range -25 ..0 .25 is displayed.
Methods:

void display()

- Drawing of the elements and calling the function **displayVal()**

void displayVal()

- Draws the trim values

boolean overT()

- Determines the new trim value when the mouse or finger is over the trim buttons.

String getSval()

- This method creates the telegram which has to be transmitted.

3.2.1.2.5 Class Switch

This class is used to create switches for switching binary actuators in model.
Constructor:

Switch (int r, int cx, int cy, int channel, int hdr)
Parameter:
r: Radius of the buttons

cx: Centre position of the guide slot – x-coordinate

cy: Centre position of the guide slot – y-coordinate

channel: channel of the PWM module, that controls the binary actuator.

hdr: Header has to be transmitted by the message

In addition, further variables are used which are shown in figure 9.

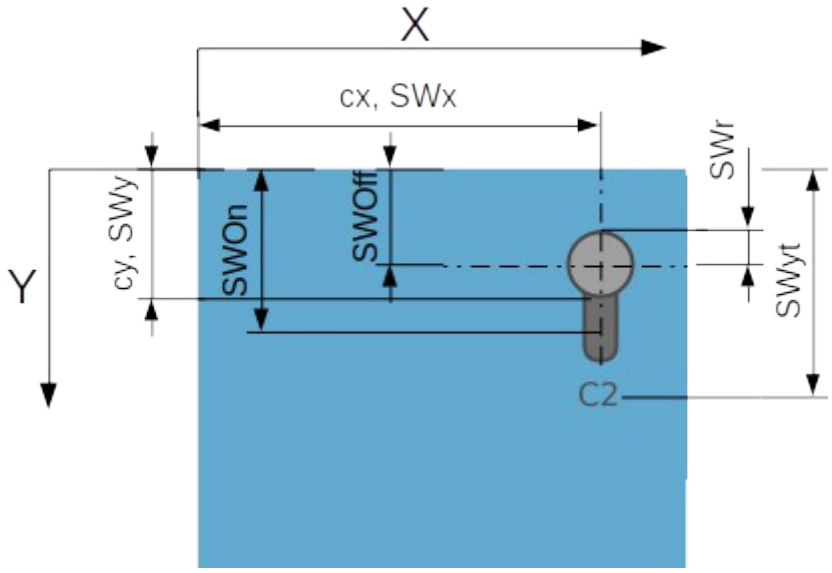


Figure 9: Switch Button Geometry

With every keystroke or touch on the area of the new button position the position is checked and redrawn.

Methods:

`void display()`

- Draws the determined elements depending on the value of the variable **`pSWPos`** in which the current value of the switch position is stored.

`void drawLabel()`

- Draws the label text

`String getSval()`

- This method creates the telegram which has to be transmitted.

`boolean overS()`

- Determines the new value of the switch position if an area has been touched that requires a change.

3.2.1.2.6 Class SwitchApp

This class inherits all properties and methods from the **`Switch`** class. It represents the switch for remote shutdown message . Only the constructor and the **`drawLabel()`** method are overwritten.

3.2.1.2.7 Class CtlBu2

This class draws two buttons that act as separate switches on one channel of the PWM controller. This is used for simple play models without continuously servo for steering.

Constructor:

`CtlBu2 (int cx, int cy, int channel)`

`Parameter:`

`cx: Centre position of the guide slot – x-coordinate`

`cy: Centre position of the guide slot – y-coordinate`

`channel: channel of the PWM module, that controls the binary actuator.`

Methods:

`void display()`

- Draws the determined elements depending on the value of the variable **`pSWPos`** in which the current value of the switch position is stored.

`String getSval()`

- This method creates the telegram which has to be transmitted.

3.2.1.2.8 Run time Behaviour and Sequence Control

The sketches of the Processing development environment have the following general structure:

- It starts with an area for defining global variables and constants. The objects are also made known here.
- **`void setup()`**
  - Definition of orientation and screen size
  - Calling the constructors of the graphical controls
  - Bind port 6000 to the IP address
- **`void draw()`**
  - This function is called with the standard frame rate of approx. 20 ms.
  - The **`draw()`** methods of all graphical control elements are called.
  - If a valid IP address has been received from the model (receiver), cyclic control data of the control lever objects are sent.
  - If there is no valid IP address, the internal variable **`ip_received`** is set to false after a timeout to prevent further transmission.
- **`void mousePressed()`** or **`void touchStarted()`**
  - Depending on the application used - PC or Android - this function is always called by the system when the mouse button is pressed or the screen is touched. It is checked whether one of the objects 'Trim' or 'Switch' was touched. As a result, the trim telegrams or switch telegrams are sent.
- **`void receive(byte[] data)`**

- This is the standard method of the UDP library, which takes over the data at the configured port and provides it to the application.
- The receiver (Rpi) cyclically sends every second an identification telegram with the following structure: ID@IP
- Example: "RC#001@192.168.43.3".
- This input is decoded and split into its components. After checking whether this is a valid IP in the local address space, it is provided to the application and the variable i**p_received** is set to **true**. In addition, the reception time is stored.

3.2.1.2.9 Adding or Removing Controls

All areas in the source code, that have to be adjusted when adding or removing elements are marked with the comment lines

```
// Start adaptation required

  ......

// End adaptation.
```

The source code was also compiled to a signed APK package. If you save this file on your mobile phone, you can easily install the application and try it out. Since I assume that many adaptation will be made for your own application, I will not publish this application in the Google Play Store.

3.2.2 Receiver (Raspberry Pi)

The receiver should process the received telegrams in such a way that servos, H-bridges and GPIOs can be controlled. Therefore we first have to deal with the interfaces of these actuators.

3.2.2.1 Servo

Analogue standard servos are controlled with a pulse of 1.5 ms +/- 0.5 ms, which is repeated every 20 ms (50 Hz). Depending on the manufacturer, there may be slight differences. Figure 10 shows the pulse at the input of a servo as it must be generated with the PWM module. The servo's angle resulting from this pulse is not standardised, but usually results in an changed angle of +/- 45 °. In our case an decimal input of 0..254 at the user interface is converted into a pulse with a width of 1.0 ms to 2.0 ms. The value 127 represents the home position of the servo.

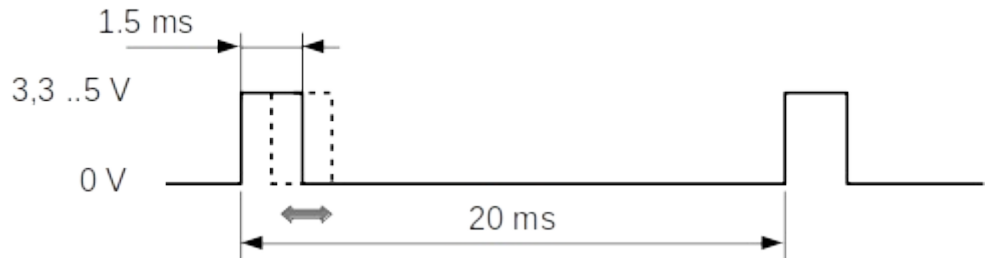
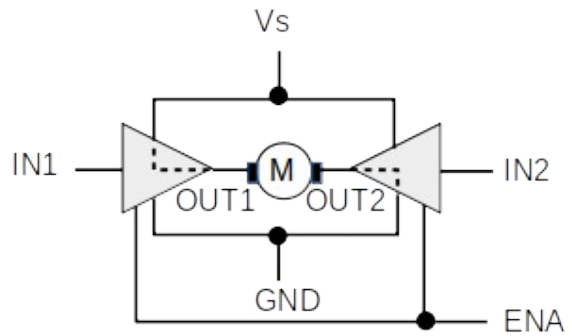


Figure 10: Impulses at the interface of an proportional Servo

3.2.2.2 H-Bridge

The hardware of an H-Bridge is explained in detail in the appendix section 'H-Bridge with LM298N'. The principle is shown in Figure 11. With the inputs IN1 and IN2 the direction of rotation can be set. ENA is used as PWM input (0..100%).



ENA	IN1	IN2	OUT1	OUT2	Remark
H	H	L	VS	GND	Forward
H	L	H	GND	VS	Backward
H	H	H	Vs	Vs	Stop
H	L	L	GND	GND	Stop
L	@	@	GND	GND	Stop

Figure 11: H-Bridge - Basic Principle

The connection of a H-bridge to the PWM module therefore requires three channels of the PWM module. With the same data as servos the driver has to supply above mentioned three outputs (ENA, IN1 and IN2).

3.2.2.3 PCA9685 Device Driver

Almost every board manufacturer like [Adafruit] or [ABElectronics] provides Python libraries in their shops. This may be enough for your first steps. But, if you want to achieve short latency in model building, you should take a close look at them. In order to avoid malfunctions, an huge number of plausibility checks are often implemented for a specific application. If you are sure that a calling function does not violate the value range of the function to be executed, you can do without it (close system). In addition, several transformations of parameter ranges are required in our case. These libraries calculates with every cycle the values. This increases the latency time. It is better to use look up tables for this value range transformation. These are calculated only once and replace the calculation at run time by simple indexing of a table. After a closer study of the PCA9685 data sheet, I noticed that the above mentioned libraries do not take full advantage of the possibilities. For example it is possible to implement a normal digital output (ON/OFF) by setting a special bit per channel.

The driver module pca9685.py provides the class **`PCA9685`**. In this class basic methods are implemented, which are based on the [Adafruit] library regarding the method and constant names. In addition, the methods specific to our application are implemented in the class **`PWM_Controller`** which provides the interfaces to the user.

3.2.2.3.1 Class PCA9685

The PWM module 'PCA9685' can set the pulse per cycle with a step size of 0..4095 (12 bit). Two

registers must be assigned with values per channel. An ON value at which the pulse jumps to high level and an OFF value at which the pulse jumps back to low level.
The constructor does not need any parameters. Since some checks have been removed, the general function of the I2C interface and the address of the PWM board must be checked before use.
The parameters ‘on’ and ‘off’ passed in the basic functions have a value range of 0..4095 and the parameter `chnl` the value range of 0..15. Compared to the ‘Adafruit’ library [Adafruit] the method `set_dio(chnl, state)` was added and the Python library 'smb' was used to control the I2C bus. The method `set_dio(chnl, state)` allows the static modification of a PWM channel output.

3.2.2.4 RCAPP

In the module ‘rcapp.py’ are implemented 4 classes.

`class Utility():`	Helper functions as static methods
`class PWM_Controller():`	User interface to the PWM controller with value transformations and parameter storage
`class Observer(Thread):`	Timeout supervision of communication
`class UDP_Client(Thread):`	Receiving and sending UDP telegrams

3.2.2.4.1 Class Utility

In this class static auxiliary methods are implemented to determine the own IP and to generate the Broadcast IP based on it. This is required to make the IP and the ID string of the receiver known in a cyclic telegram to a transmitter in the local network.

3.2.2.4.2 Class PWM_Controller()

This class represents the interface to the imported class PCA9685.
The constructor calculates with the input parameter for each channel the default pulse parameter in a two dimension table. This is done with the method ‘`calc_puls_table(self)`’ . It is possible to adjust the behaviour of each channel with the method ‘`adjustCh (self, min_val, max_val, ch)`’. The values for the timing of each channel are stored in table `‘self.ChTidef = [],[]’` for each channel with the ‘min’ value and the ‘diff’ value.

```
def __init__(self, PCA, i_min, i_max, freq, L298s = [], Dios=[], Inv = [])
      PCA : imported class PCA9685
      i_min: minimum pulse duration for servos [ms]
      i_max: maximum pulse duration for servos [ms]
      freq: Repetition rate of pulses in [Hz] (20 ms = 50 Hz)
      L298s: List of PWM channels with ENA, IN1 and IN2 input
      Dios: List of channels, which are used as switches (On/Off)
      Inv:  List of channels, with inverted control
```

At first some min. and max. values are calculated. If the ON value is set to '0', the OFF value for a certain pulse width is calculated using the following formula.

```
I_Off =  round ((ts * f)/1000) * 4095
I_Off : Value for the OFF register(0..4095)
ts    : Impulse duration
f     : Cyclic frequency of the impulses
```

The constructor of the class 'PWM_Controller', calculates default values for the fail-save function, the impulse duration for each channel and the ‘off’ impulse value for the **PCA9685** register.

```
self.imp_tab = []     - 16 x 254 cells table, containing the
                        impulse parameter for each servo value of channel
                        used by the PCA9685 device driver
self.FSave = []       - Stores the fail-save servo value of each channel
self.ChTidef = [],[] - Stores the min and diff = max - min values
                        of pulse for each channel. Please see also figure 12.
```

At the end of the initialisation run, a two-dimensional table is calculated which generates a list with a length of 255 elements for each PWM channel (0..15) in which the OFF pulse values are stored. If an H-bridge is configured with the parameter list **L298Channels**, the corresponding PWM channel is overwritten with the method **calc_puls_table_L298()** in order to cover the range 0..100%. Figure 12 shows how the value range of the look up table is derived.

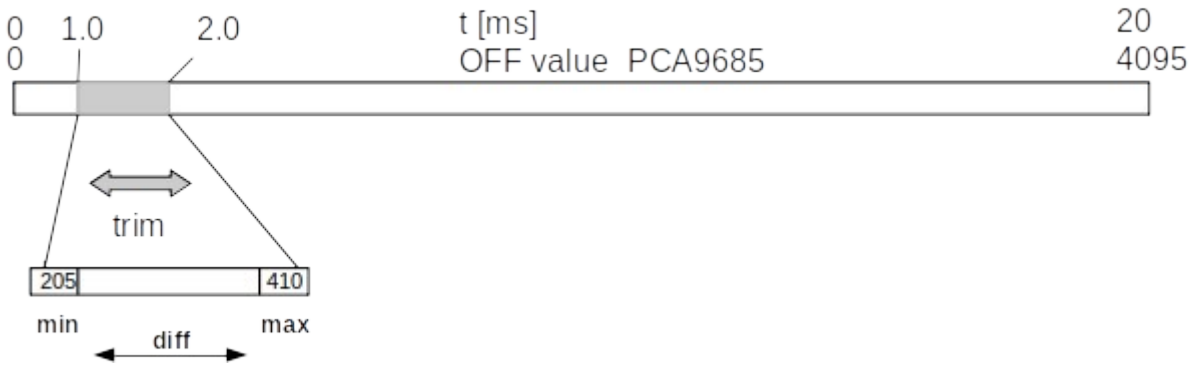


Figure 12: Creating of the ‘imp_tab’ for an variation of 1,0 to 2,0 ms

The methods that represent the user interface are described below.

update(self, msg):

This method selects in a loop from the received telegram the channel identifier and the output value for each control telegram (3 bytes) and calls the methods **update_ch**,**trimm_chan** or **update_app** depending on the header information. In addition, a time stamp is stored which is used for timeout monitoring.

update_ch(self, chan, remote_inp):

Parameter:

chan : 0 to 15 (channels 0..15 of the PWM module),

remote_inp : 0..254 (value range of the set value)

This method checks the contents of additional configuration lists. Depending on it's existence special methods for servos, H-bridges, switches are called. If channels are configured inverted, the output value of the range 0..254 is converted to the value range 254..0 with a look up table.

`trimm_chan (self, chan, trimm):`
Parameter:

chan : 0 to 15 (channels 0..15 of the PWM module),
trim : 0 to 50 (Trim value, about 20 % of the complete value range)

This method is called event driven after changing the trim values and leads to an update of the look up table of the desired channel. See Figure 11.

`update_app (self, chan, remote_inp):`
Parameter:

chan ; 0 to 15 (channels 0..15 of the PWM module) not relevant,
remote_inp: 0 = shutdown the receiver,
254 = being alive

`fail_safe(self):`
This method sets all actuators to a basic state defined by the method. Servos in a predefined position, switches in position 'Off' and the speed controller in state 'no drive'. Of course this can be adjusted. With an electric boat you might wish that it tries to reach the shore within a large radius.

Method for Configuration of the Receiver
These methods can be called before the Objects UDP_Client and Observer will be instanced.
`set_fail_save_pos (self, chan, val):`

Overrides a default value of fail-save of an certain channel in the range of 0..254.

`adjust_channel (self, min_val, max_val, ch):`

Overrides the default min. and max. impulse duration [ms] for a certain channel.
That is like a pre trimming.

3.2.2.4.3 Class Observer(Thread):

This class uses the default **`run()`** method to monitor whether telegrams are received cyclically. If this is not the case, the **`fail_save`** method is called after a timeout. In addition, further actions can be executed. After testing and trial, the Raspberry Pi can be shut down. Then a self-sufficient operation is possible. However, the prerequisite is that the application is started autonomously after the Raspberry has been switched on.

3.2.2.4.4 Class UDP_Client(Thread):

This class receives and decodes the UDP telegrams. If there is a valid telegram, the

`update_Controller(msg)` method of the `Servo_Controller` class is called after decoding. In addition cyclically every second a telegram with the own ID and the IP is sent to all participants of the local network.

3.2.2.5 Preparation of the Raspberry Pi

There are countless instructions on this topic on the Internet. Here now the essential activities, which are prerequisites.

3.2.2.5.1 Materials

Following materials are needed:

- Raspberry Pi Zero WH
- 8 GB SD-Card class 10
- USB power supply 5V 2 A or power bank 5V 2A
- USB-Cable
- 12 Bit 16 channel PWM Module (e.g. https://www.adafruit.com/product/3416)
  This module is also available from PIMORONI.
- analogues standard servo

3.2.2.5.2 SD Card-Image Preparation

Download the 'Raspbian Stretch Lite' image from

`https://www.raspberrypi.org/downloads/raspbian/`

Load the image onto the SD card with Etcher (Linux), Win 32 Disk Imager (Windows). Please do not plug it into the Raspberry immediately. For a headless operation without additional monitor and keyboard two files have to be stored on the SD card. After installing of the image, remove the SD card from the computer and insert it again. The partitions `'boot'` and `'root'` will be mounted. Create an empty text file named `'ssh'` without extension and save it on the 'boot' partition, not to confuse it with the `'boot'` directory on the `'root'` partition. Since the Raspberry Pi Zero W can only be accessed via WLAN, we have to configure the DHCP for the local network. Therefore a file named `'wpa_supplicant.conf'` has to be created. This file must have the following content:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
  ssid="[SSID_Netzwerk_b]"
  psk="[Passwort_Netzwerk_b]"
  id_str="netzwerk_b"
}

network={
```

```
  ssid="[SSID_Netzwerk_a]"
  psk="[Passwort_Netzwerk_a]"
  id_str="netzwerk_a"
}
```

Several WLAN networks can be entered as shown above. The Raspberry tries to log on to a network in the order entered. Therefore, switch off WLAN networks that are not desired but are in the file (e.g.@home WLAN).
If that is done, plug the SD card into the Raspberry and switch it on.
The following hints refer to a LINUX computer. After opening a terminal, you can determine the IP address of the Raspberry with the following command. The first three blocks of the IP must of course match with the IP of the your own network.

```
sudo nmap -sP 192.168.0.0/24
```

With the command '`ssh -l pi 192.168.0.3`' (Example) you can get access to the Raspberry Pi.
Now it is time to make some updates.

```
sudo apt-get update
sudo rpi-update
sudo raspi-config
```

The command '`raspi-config`' opens a configuration menu. With the sub menu 'interfaces' the I2C interface has to be activated.
For a comfortable working you have to install the 'Midnight Commander'.

```
sudo apt-get install mc
```

Following libraries has to be installed for the I2C bus:

```
sudo apt-get install python-smbus
```

```
sudo apt-get install i2c-tools
```

After mounting the PWM module on the Raspberry, the following command can be used to test whether the module has been recognised and which address is used.

```
sudo i2cdetect -y 1
```

Parameter 1 stands for all newer Raspberries which are delivered since 2012.
After calling the command you should see the following screen shot.

```
pi@RemoteControl:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

The default address of the module is 0x40.

Do not forget changing the default password with:

```
sudo passwd
```

3.2.2.5.3 First Test

Firstly load the three programme files `pca9685.py`, `rcapp.py` and `rcmain.py` into one directory in `home/pi`. The Midnight Commander (mc) on my PC transfer with an SFTP connection the files to the Raspberry Pi. Also under Windows there are certainly adequate tools.

The configuration of the receiver is separated in the `rcmain.py` file.

A final configuration determines to which PWM channels the actuators are connected. The marked parts of the main function have to be adjusted in `the rcmain.py file`.

```
Example Configuration of prototype (Figure 2)


def main():
    if not SIM:
        time.sleep(10)

    # determines whether a module with the L298 chip is used as an H-Bridge.
    # Channel numbers for [ENA, IN1, IN2] otherwise let the list empty.
    L298Channels = [15, 13, 14]

    # List of channels that act as a switch
    DIOs = []

    # If a servo rotates in the wrong direction or if the digital output shall
    # inverted the following list must be filled in.
    Inverted = [15]


    # Determining the minimums, maximum pulse width and frequency
    SC = PWM_Controller(1.0, 2.0, 50, L298Channels, DIOs, Inverted)
    # Now any adjustment could be done
    # Configure the min. and max. impulse duration for a certain channel
    SC.adjust_channel(0.95, 1.95, 0)
    # Setting all actuators into the fail-save position
    SC.fail_safe()
    # Specifies the port_tx and port_Rx, number of broadcasts at startup
    S = UDP_Client(SC,'', 6000, 6100, 10, "RC#001")
    # Determining of timeout [s] and the own recipient ID
```

```
O = Observer(SC, 30.0, "RC#001")
```

Now you can start the Handy application or PC application. After switching on the receiver you should see the ID in the upper right corner after about 10 seconds. The transmitter has received the IP from the receiver and can now control the actuators.
Of course you don't want to connect to the Raspberry Pi by '**ssh**' every time to start the application. An automatic start after booting is possible with the service 'systemctl':
First you have to create a file named **myrc.service**.

/lib/systemd/system/myrc.service

Content

```
-------------
[Unit]
Description=RemoteControl

[Service]
ExecStart=/home/pi/[directory]/rcmain.py
StandardOutput=null

[Install]
WantedBy=multi-user.target
Alias=myrc.service
----------
```

System service update

sudo systemctl enable myrc.service

und **sudo systemctl daemon-reload**

Now you can query or change the status with the following calls.

sudo systemctl start myrc.service

sudo systemctl status myrc.service

sudo systemctl stop myrc.service

After rebooting the application **rcmain.py** should be automatically started. You can check it with **sudo systemctl status myrc.service** after logging in via 'ssh' or it is recognised by the reaction of the system. The Raspberry Pi has to be secure switched off. That is carried out after a timeout when the transmitter has been switched off or by the special button on the samrtphone application.

4 Summary

A semiprofessional approach for radio frequency remote control of models could be demonstrated by quite simple means. This was possible in a time of 4 weeks without knowing the 'processing'

development environment. Practical experiences are still pending. But the whole approach has much more potential, because you can add all imaginable functions by yourself. For example, mixer functions are conceivable for aircraft models, or non-linear control behaviour. Additional autonomous functions and the transfer of videos to a mobile phone are also conceivable.

5 Changes History

Date	Changed Sections	Remarks / Changes
29.04.2019 Release. 1.1	All 2.2.1 Receiver 2.2.2 Transmitter 2.3.2 WiFi Interface Figure 5 3.2.1.2.6 Application 3.2.1.2.7 Class CtlBu2	Spelling and minor other adaptations Shutdown, Configurability Shutdown Additional header for shutdown message Figure 5 added Class ‚SwitchApp‘ added Class CtlBu2 added

6 Terms and Abbreviations

BEC	Battery Elimination Circuit
GPS	Global Position System
I2C	Serial 2 Wired Interface
LED	Light Emitter Diode
PC	Personal Computer
PWM	Puls Width Modulation
Rpi	Raspberry Pi
SFTP	SSH File Transfer Protocol
ST	‘STMicroelectronics’, http://www.st.com
UDP	User Datagram Protocol

7 List of References

[ABElectronics]	https://github.com/abelectronicsuk/ABElectronics_Python_Libraries/tree/master/
[Adafruit]	https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library
[AdaPWMBon]	https://www.adafruit.com/product/3416
[Deepl]	https://www.DeepL.com/Translator
[MiniSSC]	https://www.seetron.com/docs/ssc2mnl.pdf
[PCA9685]	Mouser Electronics, PCA9685,16-channel, 12-bit PWM Fm+ I 2 C-bus LED controller, Product data sheet, Rev. 4 — 16 April 2015
[Processing]	https://processing.org

[ProForAndr]	https://www.heise.de/make/artikel/Processing-fuer-die-Android-App-Entwicklung-nutzen-4117340.html
[ProForBeg]	https://www.heise.de/make/artikel/Processing-Programmieren-fuer-Anfaenger-4106608.html
[UDPLib]	http://ubaa.net/shared/processing/udp/

8 Attachment

8.1 H-Bridge with LM298N

Some boards from various manufacturers are available, which contain a double H-Bridge with the circuit LM298N from ST (see Figure 13). The design is simple. Almost all connectors of the LM298N are available. In addition, a linear 5 VDC voltage regulator is integrated. A separate 5 V voltage is only required for higher motor voltages. Then the '5V ENA' jumper must be removed. The numbering used in Figure 11 matches the PIN numbers of the LM298N data sheet. If loads with a current greater than 2 A are to be controlled, both channels A and B can be connected in parallel.

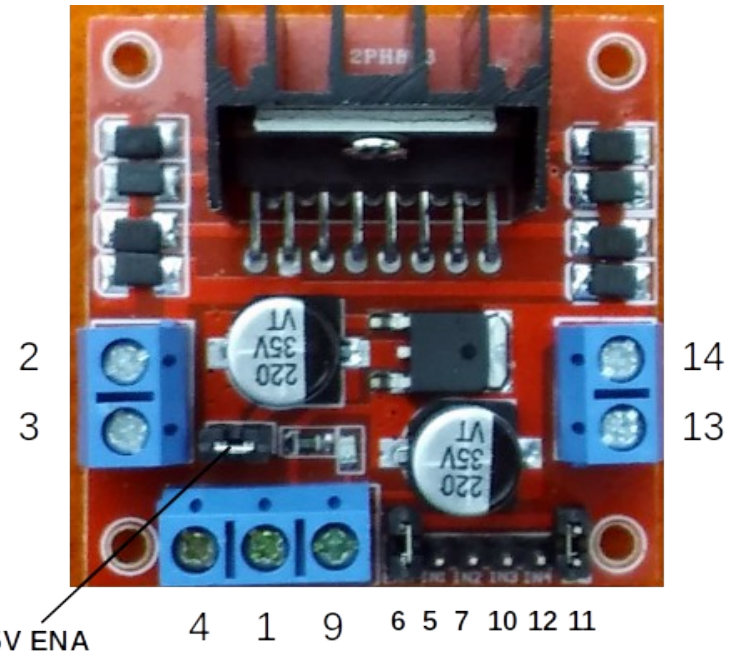


Figure 13: Motor Controller with LM298M

1	n. A.	9	Vss
2	OUT1	10	IN3
3	OUT2	11	EN B
4	V+ M	12	IN4
5	IN1	13	OUT3
6	EN A	14	OUT4
7	IN2	15	n.A.
8	GND		

Table 1 : Pin assignment