REMOTELY CONTROLLED CAR

Abstract

This documents contains the steps followed to build the keyboard controlled car project using Raspberry Pi 3 B+ and Linux Buildroot.

Supervised By

George Emad

Mohammed Tarek

Asmaa Elsayed

Asmaa.elsayed29@gmail.com

Table of Contents

Introduction 2

First Phase 2

Second Phase 9

Third Phase 10

References 12

Introduction

The purpose of this project is to integrate the concepts of Linux buildroot and Raspberry Pi board into a small embedded system that consists of a 2 wheeled car and an ultrasonic sensor that is remotely controlled using a PC keyboard.
The project consisted of three main phases; configuring the Raspberry Pi wifi, implementing the codes, and finally, integrating everything with the car body. These stages will be discussed thoroughly in three sections.

First Phase

WIFI Enabling

First phase was to start the hardware wifi on the Rpi. The first attempt was to enable WIFI as a client by logging onto an external hotspot for both the Rpi and the PC. The steps were as follows:

1. Enable these configurations on menuconfig
   a. System configuration -> /dev management -> Dynamic using devtmpfs + mdev
   b. Target packages -> Hardware handling -> Firmware -> rpi-wifi-firmware
   c. Target packages -> Networking applications -> wpa_supplicant
   d. Target packages -> Networking applications -> wpa_supplicant -> Enable nl80211 support
   e. Target packages -> Networking applications -> wpa_supplicant -> Install wpa_passphrase binary

2. Edit files to be later added on the overlay to configure the Rpi as a client and specify the name of the network it will log onto
   a. Create a file called interfaces under the directory "buildroot/board/raspberrypi/" with the following content:

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
  pre-up /etc/network/nfs_check
wait-delay 15
```

```
auto wlan0
face wlan0 inet dhcp
pre-up wpa_supplicant -B -Dnl80211 -iwlan0  c/etc/wpa_supplicant.conf
post-down killall -q wpa_supplicant
wait-delay 15
iface default inet dhcp
```

b. Create a file called wpa_supplicant.conf under the directory "buildroot/board/raspberrypi/" with the following content

```
network={
        ssid="SSID"
        psk=XXX
      }
```

c. For the scripts to be copied to the Rpi, the post build script must be modified

```
cp package/busybox/S10mdev ${TARGET_DIR}/etc/init.d/S10mdev
chmod 755 ${TARGET_DIR}/etc/init.d/S10mdev
cp package/busybox/mdev.conf ${TARGET_DIR}/etc/mdev.conf

cp board/raspberrypi3/interfaces ${TARGET_DIR}/etc/network/interfaces
cp board/raspberrypi3/wpa_supplicant.conf
${TARGET_DIR}/etc/wpa_supplicant.conf
```

The second attempt was to make the Rpi itself as an Access point with its own WIFI that the PC can log onto. The steps were as follows:

1. Enable these configurations in menuconfig
   a. Build options -> RELRO Protection -> Partial
   b. Target packages -> BusyBox -> Show packages that are also provided by busybox.
   c. Target packages -> Networking applications -> dhcp (ISC).
   d. Target packages -> Networking applications -> dhcp (ISC) -> dhcp server.
   e. Target packages -> Networking applications -> dhcp (ISC) -> dhcp server -> Enable delayed ACK feature.
   f. Target packages -> Networking applications -> dhcp (ISC) -> dhcp relay.

g. Target packages -> Networking applications -> dhcp (ISC) -> dhcp client.
h. Target packages -> Networking applications -> dhcpcd
i. Target packages -> Networking applications -> iptables.
j. Target packages -> Networking applications -> iptables -> bpfc and nfsynproxy.
k. Target packages -> Networking applications -> iptables -> nftables compat.
l. Target packages -> Networking applications -> wpa_supplicant -> Enable AP mode.

2. Edit files to be later added on the overlay to configure the Rpi as a client and specify the name of the network it will log onto
   a. Create a file called interfaces under the directory "buildroot/board/raspberrypi/" with the following content

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
  pre-up /etc/network/nfs_check
    wait-delay 15

 auto wlan0
iface wlan0 inet static
address 192.168.2.1
netmask 255.255.255.0
network 192.168.2.0
gateway 192.168.2.1
pre-up wpa_supplicant -B -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf
post-down killall -q wpa_supplicant
wait-delay 15

iface default inet dhcp
```

b. Create a file called wpa_supplicant.conf under the directory "buildroot/board/raspberrypi/" with the following content

```
network={
        ssid="RPi3B"
         mode=2
          proto=RSN
          key_mgmt=WPA-PSK
          pairwise=CCMP TKIP
          group=CCMP TKIP
           psk="12345678"
         }
```

c. Modify the file called dhcpd..conf under the directory "buildroot/board/raspberrypi/" and uncomment some lines

```
#
# Sample configuration file for ISC dhcpd for Debian
#
# $Id: dhcpd.conf,v 1.1.1.1 2002/05/21 00:07:44 peloy Exp $
#

# The ddns-updates-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
ddns-update-style none;

# option definitions common to all supported networks...
#option domain-name "example.org";
#option domain-name-servers ns1.example.org, ns2.example.org;

default-lease-time 600;
max-lease-time 7200;

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
authoritative;

# Use this to send dhcp log messages to a different log file (you also
```

```
# have to hack syslog.conf to complete the redirection).
log-facility local7;

# No service will be given on this subnet, but declaring it helps the
# DHCP server to understand the network topology.

#subnet 10.152.187.0 netmask 255.255.255.0 {
#}

# This is a very basic subnet declaration.

#subnet 10.254.239.0 netmask 255.255.255.224 {
#  range 10.254.239.10 10.254.239.20;
#  option routers rtr-239-0-1.example.org, rtr-239-0-2.example.org;
#}

# This declaration allows BOOTP clients to get dynamic addresses,
# which we don't really recommend.

#subnet 10.254.239.32 netmask 255.255.255.224 {
#  range dynamic-bootp 10.254.239.40 10.254.239.60;
#  option broadcast-address 10.254.239.31;
#  option routers rtr-239-32-1.example.org;
#}

# A slightly different configuration for an internal subnet.
#subnet 10.5.5.0 netmask 255.255.255.224 {
#  range 10.5.5.26 10.5.5.30;
#  option domain-name-servers ns1.internal.example.org;
#  option domain-name "internal.example.org";
#  option routers 10.5.5.1;
#  option broadcast-address 10.5.5.31;
#  default-lease-time 600;
#  max-lease-time 7200;
#}

# Hosts which require special configuration options can be listed in
# host statements.   If no address is specified, the address will be
# allocated dynamically (if possible), but the host-specific information
# will still come from the host declaration.
```

```
#host passacaglia {
#  hardware ethernet 0:0:c0:5d:bd:95;
#  filename "vmunix.passacaglia";
#  server-name "toccata.fugue.com";
#}

# Fixed IP addresses can also be specified for hosts.   These addresses
# should not also be listed as being available for dynamic assignment.
# Hosts for which fixed IP addresses have been specified can boot using
# BOOTP or DHCP.   Hosts for which no fixed address is specified can only
# be booted with DHCP, unless there is an address range on the subnet
# to which a BOOTP client is connected which has the dynamic-bootp flag
# set.
#host fantasia {
#  hardware ethernet 08:00:07:26:c0:a5;
#  fixed-address fantasia.fugue.com;
#}

# You can declare a class of clients and then do address allocation
# based on that.   The example below shows a case where all clients
# in a certain class get addresses on the 10.17.224/24 subnet, and all
# other clients get addresses on the 10.0.29/24 subnet.

#class "foo" {
#  match if substring (option vendor-class-identifier, 0, 4) = "SUNW";
#}

#shared-network 224-29 {
#  subnet 10.17.224.0 netmask 255.255.255.0 {
#    option routers rtr-224.example.org;
#  }
#  subnet 10.0.29.0 netmask 255.255.255.0 {
#    option routers rtr-29.example.org;
#  }
#  pool {
#    allow members of "foo";
#    range 10.17.224.10 10.17.224.250;
#  }
#  pool {
```

```
#    deny members of "foo";
#    range 10.0.29.10 10.0.29.230;
#  }
#}

subnet 192.168.2.0 netmask 255.255.255.0 {
  range 192.168.2.10 192.168.2.50;
  option broadcast-address 192.168.2.255;
  option routers 192.168.2.1;
  default-lease-time 600;
  max-lease-time 7200;
  option domain-name "local";
  option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

d. The above created files must be copied, so add the following lines to buildroot/board/raspberrypi/post-build.sh

```
cp package/busybox/S10mdev  ${TARGET_DIR}/etc/init.d/S10mdev
chmod 755 ${TARGET_DIR}/etc/init.d/S10mdev
cp package/busybox/mdev.conf ${TARGET_DIR}/etc/mdev.conf

cp board/raspberrypi3/interfaces ${TARGET_DIR}/etc/network/interfaces
cp board/raspberrypi3/wpa_supplicant.conf
${TARGET_DIR}/etc/wpa_supplicant.conf
cp board/raspberrypi3/dhcpd.conf ${TARGET_DIR}/etc/dhcp/dhcpd.conf
```

e. Create a file, named sysctl.conf, and copy to /etc/ on the root partition of the memory card:

```
# Enable IP forwarding.
net.ipv4.ip_forward = 1
```

The scripts in /etc/init.d/ run after each boot and before each shutdown. The following script is responsible for setting the sysctls above. So create another, **executable** file, named `S02procps`, and copy to /etc/init.d/ on the root partition of the memory card:

```
#! /bin/sh
if [ "$1" == "start" ]; then
   sysctl -p
fi
```

f. Create another **executable** file, named S99firewall, and copy to /etc/init.d/ on the root partition of the memory card:

```
#! /bin/sh
if [ "$1" == "start" ]; then
   iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
   iptables -P FORWARD DROP
   iptables -A FORWARD -i eth0 -o wlan0 -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
   iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
   iptables -P INPUT DROP
   iptables -A INPUT -i eth0 -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
   iptables -A INPUT -i wlan0 -j ACCEPT
fi
```

Second Phase

Writing Code

Second phase was to start implementing python codes to move the motors, write code for ultrasonic calculations, and to take input from the keyboard to start controlling the car. First, we must install the python package following these steps

1. Download the python package for the Rpi from menuconfig
   a. Select BR2_PACKAGE_PYTHON
   b. Select BR2_USE_WCHAR
   c. Select BR2_TOOLCHAIN_HAS_THREADS
   d. Deselect BR2_STATIC_LIBS

2. Download WiringPi library to be able to use the GPIO of the Rpi using its functions

a. Select **wiringpi** then **make**

- First problem to encounter was that the library couldn't be included even after It was made sure that it was downloaded to the Rpi.

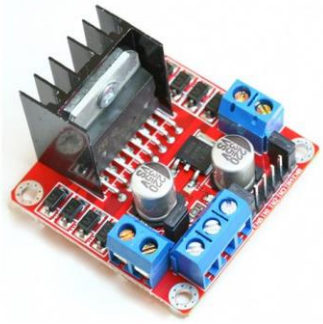
3. Download GPIO library to be able to use the GPIO of the Rpi using its functions
    a. Select **PYTHON_RPI_GPIO** then **make**
        - Second problem was that the GPIO library needed dependencies that would change the architecture of the Rpi, which after following the path lead to an error during the make which that the toolchain of the arm needed to be installed.
        - According to the timeline constraints, it was decided to cancel working with both libraries and chose to work with the normal shell commands to directly talk to the GPIO pins. This was done in conjunction with the python script.
4. Download curses library to get input from keyboard and use the inputted characters to control the motors
    a. Select **curses** then **make**

Third Phase

Hardware Interfacing

Working on the car body and figuring out which power source to use for the motor, sensors, and for the Raspberry Pi. For safety reasons, as the motor drivers might draw some high torque and current and it might affect the Rpi, it was decided to separate the power for the Rpi from the rest of the components. An H-bridge module was used as a driver for both the DC motors.

Number	Component	Picture
1	Car Chassis	

2	Ultrasonic Sensor	
3	Lithium Ion Batteries	
4	Power Bank	
5	H-bridge Module	

References

- https://blog.crysys.hu/2018/06/enabling-wifi-and-converting-the-raspberry-pi-into-a-wifi-ap/
- https://bootlin.com/doc/training/buildroot/buildroot-slides.pdf
- https://www.explainingcomputers.com/rasp_pi_robotics.html
- https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi