ROTA

Remotely Operated Transportation Apperatus

Rijksuniversiteit Groningen

Bastiaan van Houttum - S3146391
Oliver Strik - S3100693

November 2017

Contents

1 Problem Description

The idea was to make something interesting, what was decided on was a toy car that can be controlled from your phone. The device would need a way of receiving input, such as WiFi and a way of processing and acting upon input such as an Arduino.

2 Problem Analysis

In order to achieve such a complex project, a plan had to be made. The problem was broken into two steps, the first was designing the software, the second was then fitting the hardware together. The first question devised was: "Can we make a toy car drive around, and prevent the user from crashing it into objects"?

Given the fact that the expected scope of the project is relatively small, and the limited amount of time that was available, it was concluded to be best to simplify the problem. Designing and printing the body, as well as designing an anti-crash system on top of designing an app and communications system between three devices seemed infeasible. Thus the second question devised was: "Can we mod or hack an RC car such that we can control it over wifi"? This was decided to be possible with the resources that were available.

3 Design

In order to build the car, the following hardware was used:

- Arduino Uno
- Velleman Stepper Motor Controller
- Servo from the Arduino started kit
- Raspberry Pi B Model (v1.2)
- Generic WiFi USB Adapter
- 8GB Micro SD card
- RC car which would have otherwise been forgotten about
- Power bank with 2 USB ports to power the Arduino
- Jumper wires
- 9V block

The Raspberry Pi was first configured by loading the Raspbian OS on it, but this proved to be flawed when setting up a WiFi network on the Pi, since there were many complications with the network. Therefore, it was decided Arch Linux would be better suited. A server for the Raspberry Pi with which our app communicates was then written in C.

The server and app were designed such that the app sends a packet describing the position of the two joysticks, this packet consists of a total of 5 bytes. Two for each joystick and an extra one in case something needed to be added at a later date. The server receives this packet from the app, and sends it via a serial connection to the Arduino. This is then read by the Arduino, which translates the packet into the motor speed and servo direction.

At the same time, the controller app was developed. Android was decided as deployment platform, because of its expansive support documentation and general ease of use. The app

would have two joysticks, and sends a packet of 5 bytes to our server. The packet contains an undefined byte, the x and y for the left joystick and the x and y for the right.

The last piece to develop was the code for the Arduino to interpret the packet of data sent to it from the server.

Finally, it was just a matter of assembling the components. After piling all the components listed above on to an old RC car frame, the result was an odd looking Android controlled RC Car.

4 Program Code

The code of ROTA is split into three seperate programs.

- The controller - Written in Java. [1]
- The Server - Written in C. [2]
- The Interpreter - Written in Arduino C/C++. [3]

In the interest of saving trees, the 700+ lines of code are not included in this document, if you wish to find them please use the links supplied in the references. A description of each program, however, is provided below.

4.1 The Controller

The controller was written in Java with the help of the Android Studio IDE. The main function of the controller is to create two floating joysticks to allow users to submit input to ROTA. It is essentially an Xbox or PlayStation controller but without any buttons. One joystick is created on the left and one on the right, they will place themselves under the first touch on their half of the screen and the centre of the joystick will remain at that point until the finger is lifted again. This allows for some degree of personal adjustability; not everyone holds their phone the same way, and not all phones are the same size or shape, so the joysticks should not be fixed to one place on the screen.

The controller also tries to connect to a TCP socket on the IP address 192.168.12.1 port 55455, the Raspberry Pi was set up to maintain a consistent gateway address so that the controller would never have to ask for the IP address of the server, the server was also configured to only use port 55455. Once a connection has been created the controller will send an update packet every few miliseconds. This packet consists of 5 bytes {a,x1,y1,x2,y2}.

1. 'a' is an undefined byte. It can be used for anything; it was added in case some kind of status update or emergency stop button was introduced.
2. x1 and y1 are the influences of the left joystick. Their value will be between -127 to 127. (0,0) would mean the joystick is centered. (0,127) would mean the joystick was centered on the x-axis but pushed all the way up on the y-axis.
3. x2 and y2 are just the same as x1 and y1 but are for the right joystick.

The packet does not require any leading or ending byte as TCP guarantees that all packets will arrive and they will all arrive in the correct order. Thus, on the server, if the Server reads 5 bytes it will always read {a,x1,y1,x2,y2} if a packet exists to be read.

4.2 The Server

The server is by far the most convoluted program of the three, which is thanks to it having been programmed in C. It does only one thing: forward all packets received from the controller, straight to the Arduino over serial. It does, however, add 127 to each of the signed bytes as the serial connection seemed to really dislike the signed bytes. 127 is then immediately subtracted from the byte when it gets to the Arduino. This is all handled in about 5 lines of code, however, there are nearly 300 lines surrounding it most of which is simply initialisation for the TCP server and serial connection. On the Raspberry Pi, the server is added to the list of programs to start at boot with systemd, thus no login is required, the Raspberry Pi must only be booted for the server to be created.

4.3 The Interpreter

The interpreter, for lack of a better name, is the program running on the Arduino which 'interprets' the x and y influences recieved from the server and translates them into motor speed, servo angle or whatever else we happen to connect to ROTA. This job could have been done with the Raspberry Pi alone, however, we would then have had the constraints of few GPIO to communicate with, as well as the lower voltage of the GPIO (3.3V instead of 5V) to deal with. Thus we decided to use the Arduino as the step between the Raspberry Pi and the extra peripherals.

After initialising the serial connections and any global variables, the Arduino then reads a single byte from the serial buffer and acts upon it. It keeps count of which particular byte it is reading and then, as we have it currently set up, if it is the left y, uses it to set the motor speed and direction or if it is the right x, use it to set the servo angle. The particular stick or axis that controls each of these can easily be changed, the only reason we have it set up as so is that it made practical sense to split the controls of throttle and steering on to separate sticks. As for why the sticks are capable of moving in both axis when only one is required, it doesn't impair their function, it was easier to make two identical joysticks over two unique ones and most importantly, if we had the time and the servos to do it, we would have like to have added a moveable camera. Unfortunately, this was not the case, but now at least it features the ability to have such a camera with the addition of only two if statements.

5 Tests and Results

Figure 1: A photo showing the joysticks in the app.

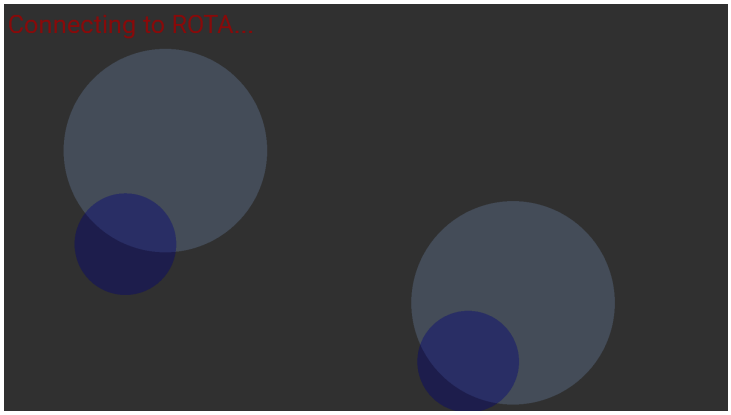

Figure 2: Another photo showing the joysticks in the app. This shows how the joysticks will move when the user touches the screen.

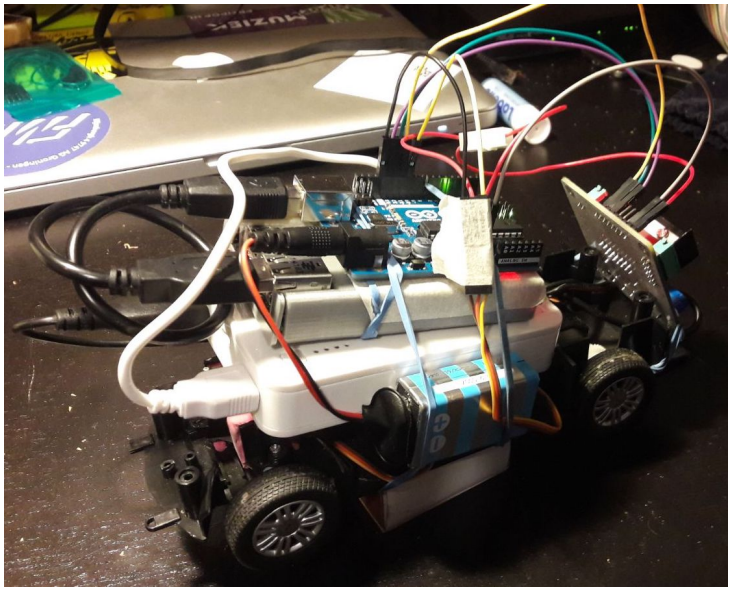

Figure 3: A view of the left side of the finished car.

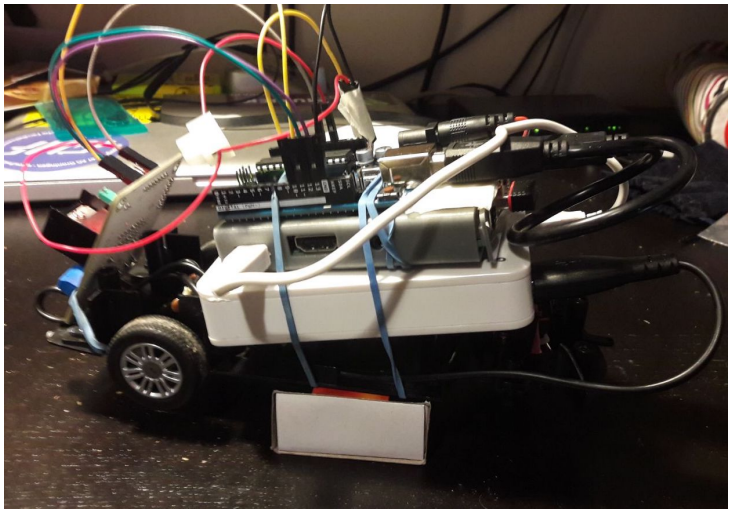

Figure 4: A view of the right side of the finished car. You may notice the missing front wheel that was being repaired at the time of this photograph.

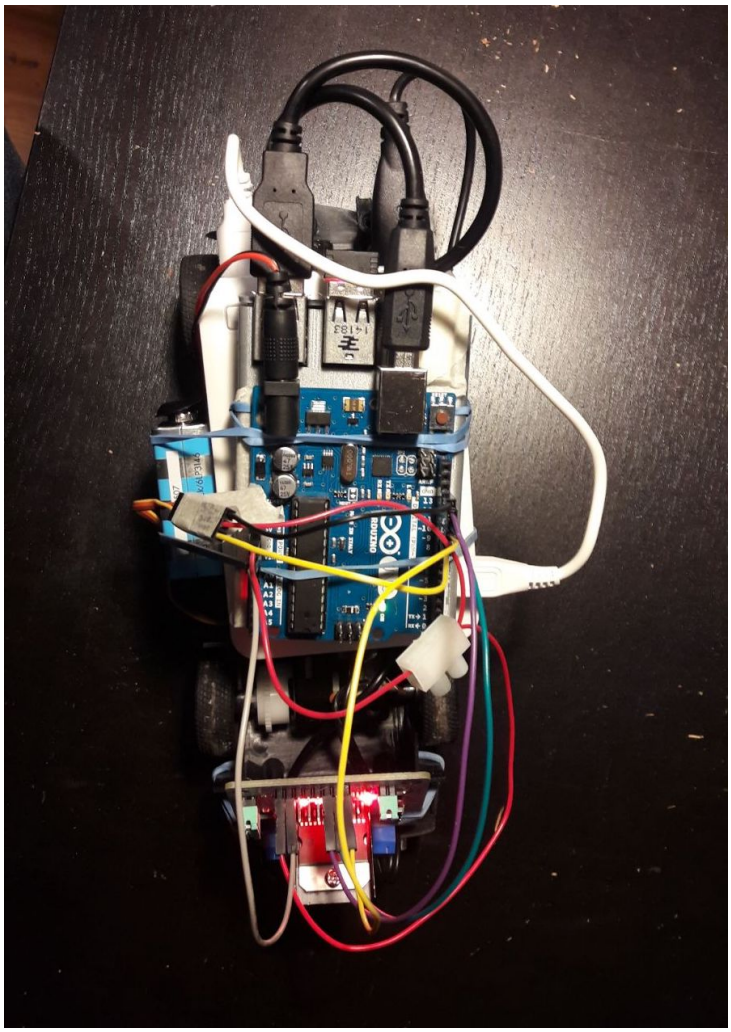

Figure 5: A view of the top of the finished car.

6 Evaluation

Throughout the development of ROTA there have been things that needed to be tested. After all, there are three devices that could fail. The first thing that was programmed was the app, after fixing many small issues that showed up throughout development, three major problems showed.

1. If the app is disconnected from the server, by losing the wifi or something similar, it would never reconnect. To fix this, the client initialisation thread was extended to perpetually loop, if the client was not connected, it would try to reconnect it, otherwise, it would send an update packet. This also helped limit the number of packets being sent to the server as originally a packet would be sent every time the user stimulated the touch screen, this helped prevent the Arduino from becomming overloaded.

2. If the screen focus shifts, causing the app to leave fullscreen, it never reverts. This does not impact usability, however it is annoying. This was fixed by giving the Android immersive environment the sticky flag. Now, popups and dragging the system bar into view does not resize the app as they are rendered 'over' the app. When they disappear the app is still fullscreen.

3. If the user leaves the app, Android will 'suspend' it. This is great, however, it wreaks havoc with the server connection. Very strange things can happen, the best you can hope for is that the app crashes, at worst, the app tries to reistablish the connection and you end up with 5 or 6 rogue threads spamming TCP Requests. To prevent this, the app was set up to destroy itself completely when minimised. This loses no functionality, however, as the app would have to be restarted when it comes out of suspension anyway, it is just easier to restart from the beginning of the program.

There were never really any issues with the server, par some minor confusion caused by C's lack of a 'byte' type and chars being unsigned on some systems but not others. This was easily fixed by declaring all chars, being used as bytes, as signed chars.

Another problem came with the Arduino, it would not accept signed bytes across serial. this meant that either the transmission method needed to be revised or that unsigned bytes needed to be sent instead. As all bytes were between -127 and 127, adding 127 before sending and subtracting it after receiving solved this problem.

The biggest problem of them all was how to supply power to the Arduino, the Raspberry Pi, and the motor shield. In the end, it was decided the Raspberry Pi and the motor shield would receive power from a large USB power bank. The Raspberry Pi would then power the Arduino. This seemed to work on the bench when the Arduino was plugged into the PC for tuning the servo and running light tests on the motor. However, the Raspberry Pi could not deliver the current to the Arduino to power the servo and the logic side of the motor controller, this would crash the Raspberry Pi and all hell would break loose. To counter this issue the Arduino was given its very own 9v battery. While this is not perfect, it does seem to solve the problem. Ideally a dedicated Lithium Polymer battery would have been preferred, a 2200mah 7.4v LiPo would be small and light enough to fit on the frame, however, they are not exactly cheap and the Raspberry Pi would require something extra to step the voltage down to 5v.

After the power problem was sorted, the car worked almost perfectly, it can pick up quite some speed, every now and then there is a slight delay in the controls however not enough to cause serious issues. The exact reason for the delay is not understood, amongst other things it could be due to the Arduino being flooded with more packets than it can handle, or it could be because the Raspberry Pi suffers from a similar problem. The only minor faults are that the servo is rather poorly attached to the steering, allowing for a small margin of free movement. This means that the steering is never exactly centered and the car needs constant steering

adjustments to stay on course. It was also noticed that the front wheels would lock on the frame when turning. This was because all the weight had crushed the suspension. To fix this the wheel arches were removed.

References

[1] Strik, O.H.P (2017). The Source Code for the ROTA Android Controller. Available on: https://www.github.com/Kranex/ROTA-Controller

[2] Strik, O.H.P (2017). The Source Code for the ROTA Server. Available on: https://www.github.com/Kranex/ROTA-Server

[3] Houttum, B.F.W.M van (2017). The Source Code for the Arduino interpreter program. Available on: https://www.github.com/NotSoScientific/ROTA-Interpreter