RF-CONTROLLED ROBOT CAR

FINAL PROJECT REPORT

ECE 380L

Real-Time Operating Systems

Submitted by:

Allen Jiang

Sophia Jiang

Department of Electrical and Computer Engineering

The University of Texas at Austin

Austin, Texas 78712

May 9, 2022

TABLE OF CONTENTS

INTRODUCTION **2**

Project Introduction 2

Bill of Materials 3

HC-12 Introduction 3

HC-12 Testing 5

DESIGN **6**

Mechanical Design 6

Hardware Design 7

Software Design 8

CONCLUSION **14**

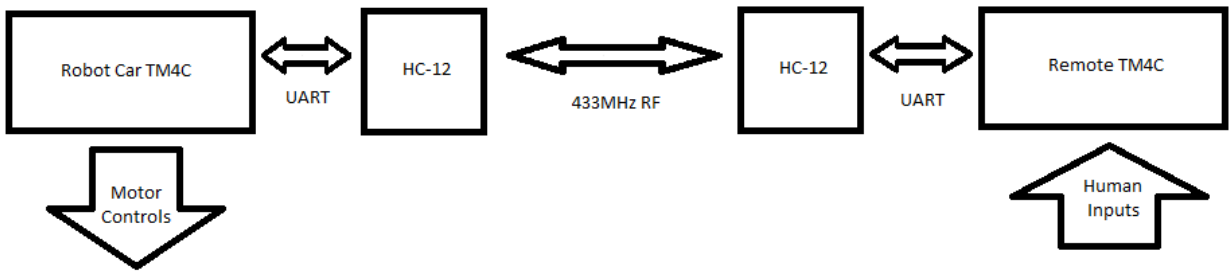
Future Recommendations 14

Conclusion 16

INTRODUCTION

Project Introduction

Our final project is an alternative application of the robot car lab, exploring the use of radio control using the HC-12 wireless transceiver module. We adapted a controller PCB from a previous project to provide human inputs to TM4C on the controller. This TM4C translates button and trigger inputs into left and right motor speeds and sends it over the 433 MHz band to an HC-12 on the robot car. The robot car sensor board TM4C relays this data to the motor board TM4C via CAN and outputs the corresponding PWM to achieve the desired speed.



Block Diagram of High-Level Overview

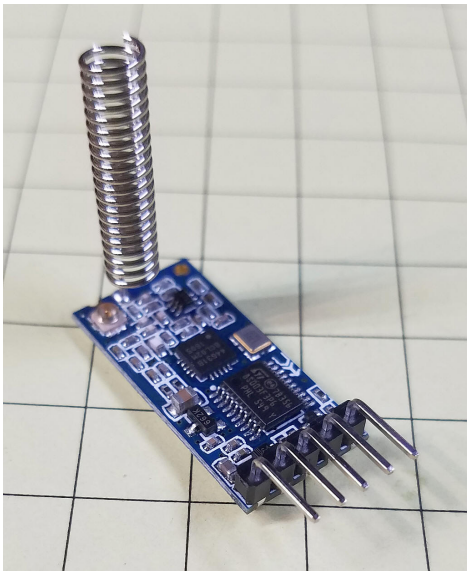
Bill of Materials

Component	Component Description	Vendor
HC-12	433Mhz Wireless Serial Transceiver	HC-Tech
Sensor Board	Sensor Board used for Robot Car with TM4C Launchpad attached	ECE 380L.12
Motor Board	Motor Board used for Robot Car with TM4C Launchpad attached	ECE 380L.12
Robot Car Chassis	Structural frame and motors	ECE 380L.12

Triggers	Finger Trigger Switch for Human Input	Digikey
Buttons	Push Switch for Human Input	Digikey
ILI9341	2.8” Color TFT LCD	Adafruit
ST7735	1.8” Color TFT LCD	Adafruit

HC-12 Introduction

The HC-12 is a sub-GHz radio frequency (RF) module that is capable of half-duplex serial communication. It uses the Si4463 transceiver, which provides wireless communication in the HC-12 module. It has a 100 mW transmitter, a receive sensitivity of -129 dBm, and can be configured to change between 100 channels within 433.4 and 473.0 MHz. The HC-12 has a transmission range of roughly 1 km with direct line-of-sight in an open area, but this range is reduced when there is interference from foliage and buildings around, impeding communication.



An Assembled HC-12 Radio Module

Onboard the HC-12 module is a STM8S003FS microcontroller. This microcontroller has been programmed with firmware to control the Si4463 and interface with another microcontroller externally using UART.

The HC-12 is interfaced using UART (RX and TX line), a +5V VCC and GND, and a command mode pin to modify module configurations. Setting the command pin to low sets the pin to command mode. Inside this command mode, the configuration of the HC-12 can be modified using AT commands sent over UART. These are the AT configurations we used:

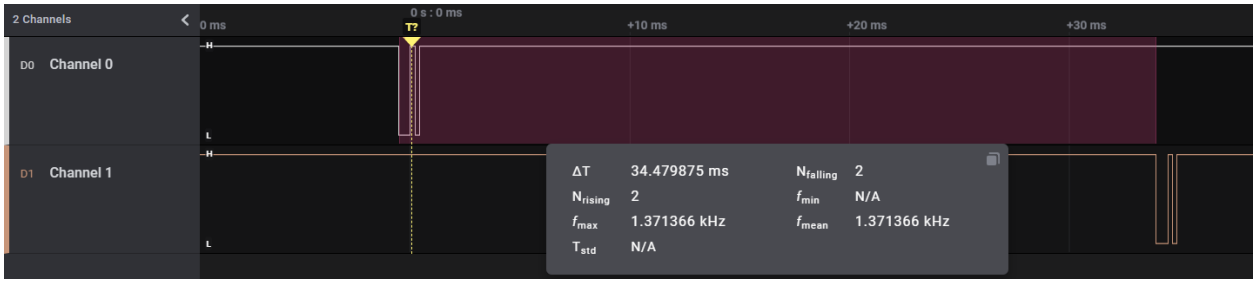
- "AT+B9600"
    - Change the serial baud rate to 9,600 bps
- "AT+C001"
    - Change wireless channel to channel 001, with a working frequency of 433.4 MHz
- "AT+FU3"
    - Set HC-12 to transmission mode 3, which configures system to full-speed mode, where the baud rate in-the-air is automatically adjusted according to the baud rate of the serial port
- "AT+P8"
    - Set transmission power to a maximum of 20 dBm
- "AT+U8N1"
    - Configure serial port format to be eight data bits, no check bit, and one stop bit

With matching configurations, two controllers will be able to use HC-12s to send data to each other by simply transmitting data to the HC-12 over UART. The data is buffered in a FIFO

in memory by the STM8S003FS microcontroller onboard the HC-12 module. The data is then transmitted over the air, and after it is received and processed by the receiving HC-12, the data is then transmitted to the receiving controller over UART.

HC-12 Testing

We tested the latency and range of the HC-12 to explore its transmission capabilities. We decided the best way to measure latency is the elapsed time between start of transmission to the start of receive. In the logic analyzer waveform below, HC-12 TX is on channel 0 and HC-12 TX is on channel 1, and the purple highlighted portion is the time difference, which was measured to be 34.48 ms.



Time between sending data on one HC-12 to receiving on the other is 34.47 ms

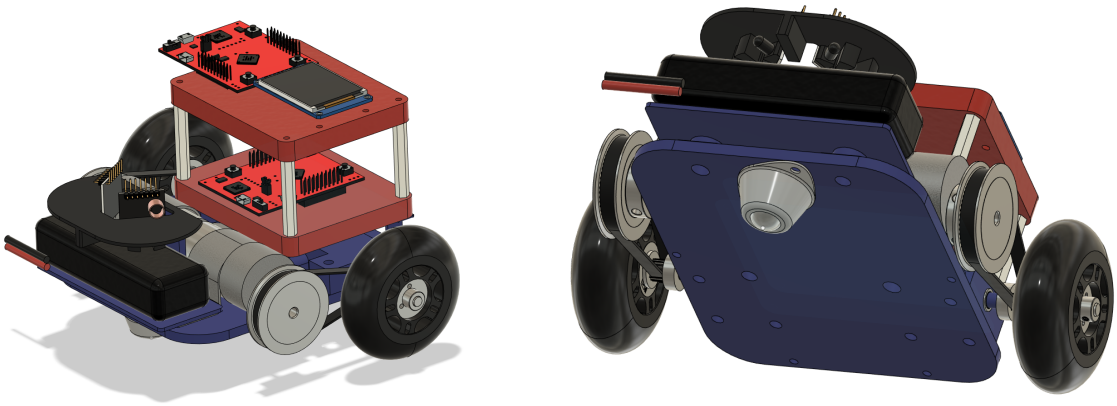
To find the range, we stood at the farthest possible distance away between the EER to CPE, which was measured to be 620 ft on Google maps. There was a small amount of packet loss when the HC-12 was blocked by a laptop screen, and no packet loss when there were no obstacles blocking it. Thus, we know that the range of the HC-12 is at least 620 ft when in a direct line of sight.

Testing distance from EER to CPE is 620 ft on Google Maps

DESIGN

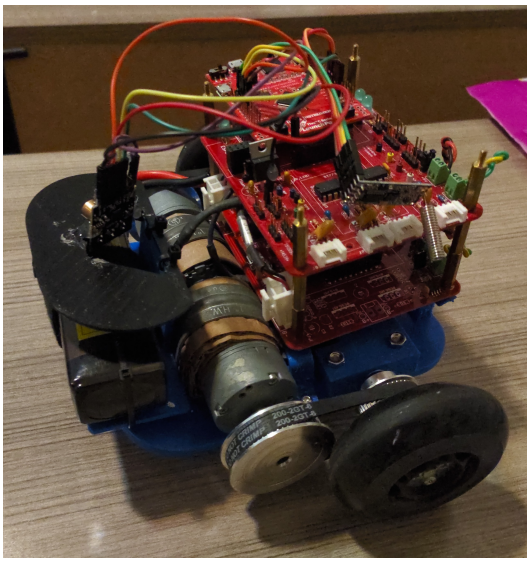
Mechanical Design

We reused our car from Lab 7 for this project. We redesigned the car frame from its original laser-cut chassis so that the car could operate on a differential drive.



3D CAD Renders of the Robot Car

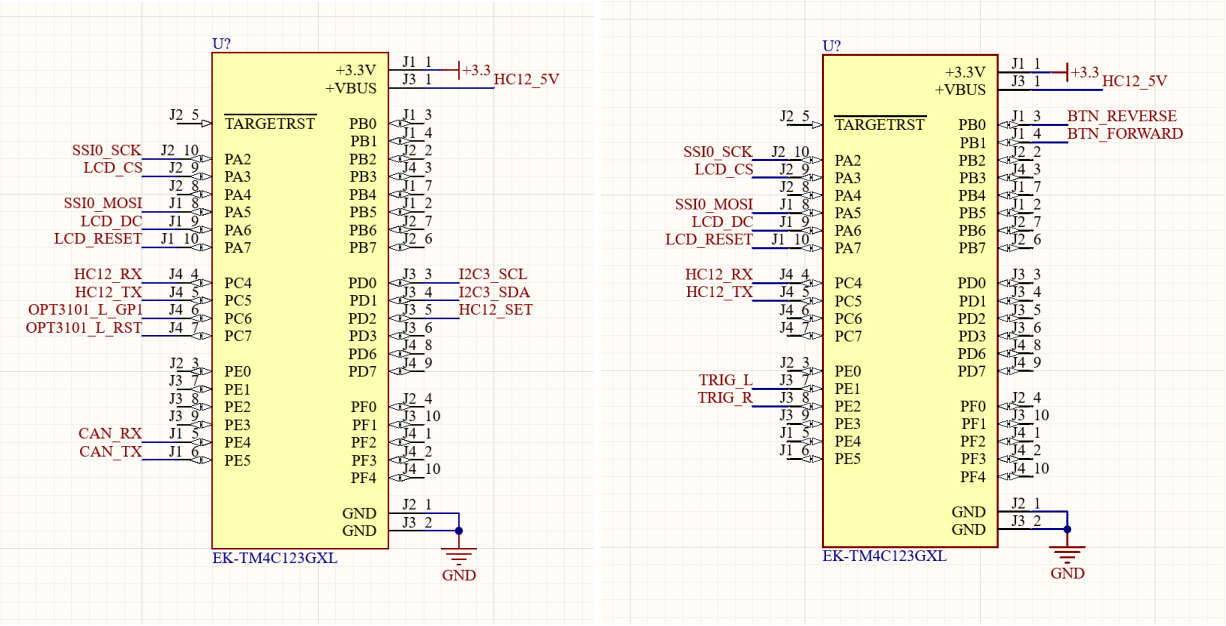
The frame is made of 3D printed parts and is assembled with parts from the provided robot car and a caster ball in front to replace the four-wheel drive.



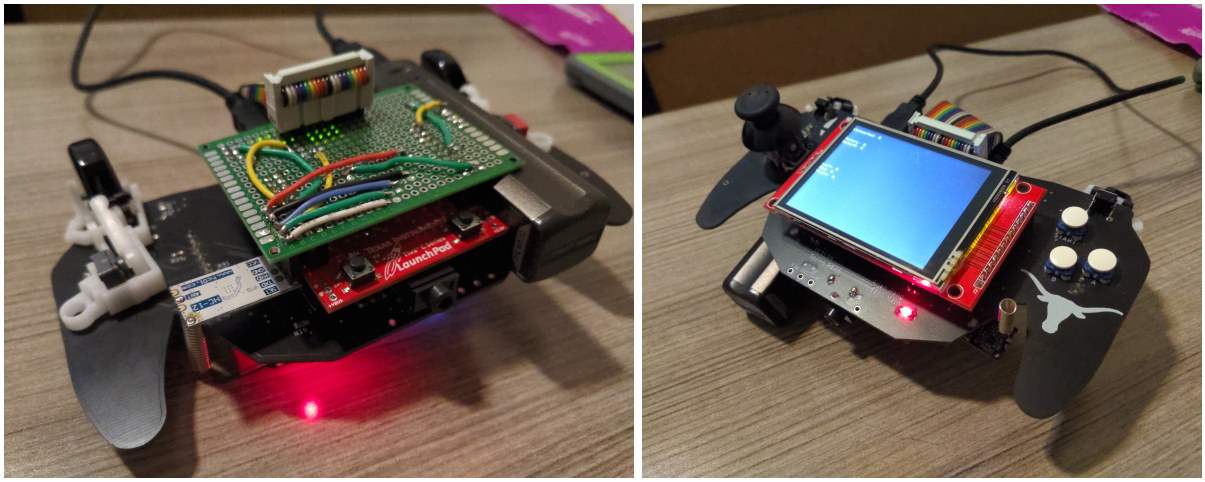
Robot car with the HC-12 module

Hardware Design

The hardware of the remote controller is reused from an old EE 319K project. This hand-held controller uses the ILI9341 LCD screen, which is similar in functionality to the ST7735 LCD. Also, there are buttons and triggers available, which are used for controlling the car. We used a prototype board to develop an extension card for the TM4C to support the connections for an HC-12 module.



Modified robot car and remote controller TM4C connections



Remote controller hardware adapted from an EE 319K project

Software Design

Our firmware for both the remote control and the robot car builds upon our RTOS from Lab 4. Using an RTOS allows us to run multiple tasks concurrently, making development easier. Being able to configure priority levels and background/foreground task contexts is helpful for accommodating different task deadlines in this real-time system. Also, multitasking increases CPU usage efficiency.

We decided to focus our communication protocol on packetized communication. This way, we could send multiple types of data at any length necessary, rather than only sending one type of data. To decide on the structure of the packet, we first had to choose between a fixed-length packet or a variable length packet. Since we were concerned about packet corruption during transmission, we chose to use a fixed-length packet. If an unknown byte of the packet was corrupted during transmission, it would no longer be possible to verify the checksum with the length of the packet. We decided on a fixed packet length of 6 bytes, including the header and the checksum. The final structure of the packet is as follows:

- Byte 0: Header

    - Describes packet type
    - Value range between [0, 15]
- Byte 1 - 4: Data
    - Contains packet data
    - Value Range between [0, 255]
- Byte 5: Checksum
    - Calculated using chained XOR of bytes 0 - 4 (longitudinal parity check)
    - Value Range between [0, 255]

When a transmission is received on the HC-12, the HC-12 communicates the transmission one byte at a time to the TM4C using UART. Our HC-12 driver takes advantage of the UART interrupt functionality on the TM4C to receive data from the HC-12. Using an interrupt saves polling time and also guarantees that all bytes are received from the UART transmission regardless of the state of the RTOS. When the HC-12 transmits a UART message to the TM4C, the UART interrupt is triggered. Inside the UART interrupt service routine, the UART message is read into a receive FIFO located in memory with a size of 100 bytes.

A task running at 50 Hz is dedicated to parsing the receive FIFO. Instead of parsing the FIFO during the background UART interrupt, we parsed the data in a foreground task context to reduce potential jitter, since parsing the receive FIFO, which requires lots of CPU cycles, would introduce greater jitter when in a background task context. This task was initialized with a higher priority than other tasks in an effort to reduce latency between receiving the packet and processing the packet.

The parsing works by first searching for a potential packet header, which is a byte between [0, 15]. With this packet location, the checksum of the data can be checked and verified. If the checksum is invalid, the firmware assumes that the packet was either corrupt or not actually a header, and the potential packet header byte is discarded and ignored. When a checksum match is found, the packet data is processed according to its header value.

The first iteration of our wireless firmware attempted to utilize the HC-12's bidirectional capability. In this iteration, each packet is sent asynchronously at a designated frequency. Here were the packets sent:

- Remote Controller to Car
    - Heartbeat (10 Hz)
    - Left and Right Motor Controls (20 Hz)
- Car to Remote Controller
    - Heartbeat (10 Hz)
    - Wall Sensor Values (10 Hz)

However, with this setup, we were not able to maintain a reliable connection and consistently receive correct packets. Some observations we noticed were that if an HC-12 transmitted a packet immediately after receiving data, that packet would never be transmitted successfully. Also, both the remote and the car would sometimes attempt to transmit a packet at the same time. When this happened, none of the HC-12s would receive anything. From these observations, we believed that the reason behind this unreliability was because the HC-12s have trouble quickly switching between receiving and transmitting data, and there may be interference with each other in the air.

In an attempt to keep the communication channel clear, we modified the protocol with the goal of ensuring that only one HC-12 is broadcasting data at any single moment. To do this, one side of the communication must be designated as the master, which directs when any communication can be done, and one side must be designated as the slave.

Since most of the essential data is sent from the remote controller to the car, the controller was designated as the master, and the car was designated as the slave. Normally, the controller would transmit data in a one-sided communication with the car, and the car would not transmit anything back. However, when the controller requires data from the car, it will transmit a packet requesting the certain data type from the car. Then, the controller will hold off on any transmissions while waiting for the car to respond with the requested packet. The controller will only continue transmissions after the requested data packet is received from the car. This way, we can guarantee that only one HC-12 will be transmitting at any point in time. Here were the packets sent:

- Remote Controller to Car
    - Heartbeat and Request Heartbeat (10 Hz)
    - Left and Right Motor Controls (20 Hz)
    - Request Sensor Data (10 Hz)
- Car to Remote Controller
    - Heartbeat Request Response
    - Sensor Data Request Response

However, there is a chance of disruption between the HC-12s during a transmission, and either the car would receive a corrupted data request packet, or the remote controller would

receive a corrupted data packet. This would cause the controller to permanently wait for the data packet after requesting it and cease communications. To resolve this issue, we added a resend request feature. When a packet has not been received in a designated time period requirement, the controller will automatically resend the request. This time period requirement is selected to be larger than the maximum latency between transmission and reception of a packet.

Even after implementing the master-slave communication, we still had issues with high packet loss and consistently receiving correct packets. Unless the delay between two different HC-12s transmitting data was greater than roughly 200ms, the transmission of the second HC-12 packet would typically be corrupt. We still do not know the reason behind this high packet loss because the hardware design of the HC-12 is still a bit of a black box, as the datasheet for the HC-12 is not detailed enough.

As a result, the bidirectional communication between the remote controller and the car was too unstable to use. Our implementation would force us to choose between high packet loss and unreliable communication or absurdly high latency. Because of this, we changed our firmware to only use one-sided communication. The controller would continuously send heartbeat and motor instructions to the car, and the car would never transmit any data back. Here was our final communication setup:

- Remote Controller to Car
    - Heartbeat (10 Hz)
    - Left and Right Motor Controls (20 Hz)
- Car to Remote Controller
    - No Transmissions

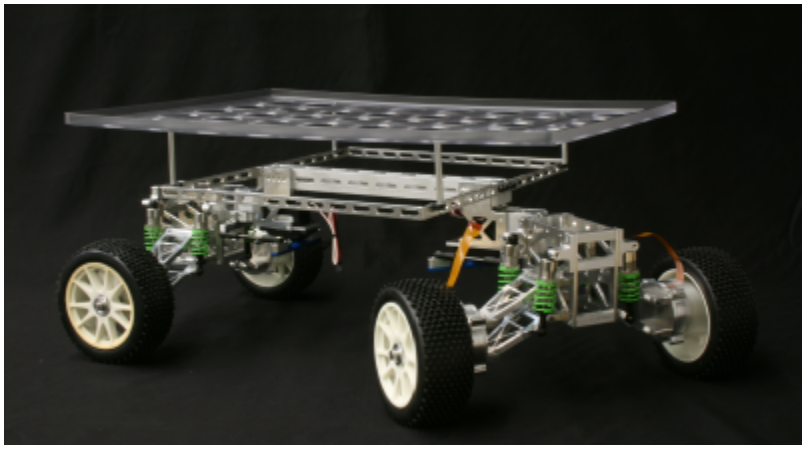
The drawbacks of this implementation meant that we would not be able to tell if we were connected to the car from the remote side, and we would also not be able to receive sensor data from the car, such as wall sensor readings from the OPT. However, the benefits were high packet transmission reliability and low latency. With this setup, we were able to reliably control the car motors.

CONCLUSION

Future Recommendations

There are a few possible future improvements we can add to this project. The most important improvement would be to figure out how to implement consistent bidirectional communication with the HC-12s, if the hardware allows. This is a very important feature to have, as bidirectional communication is necessary in many applications.

Moreover, there are various settings on the HC-12 accessible with AT commands that may be useful for certain applications. The baud rate of the HC-12 can be decreased or increased, affecting bandwidth and range. There are also different power and transmission modes to be experimented with, which can decrease power usage for low-power situations at the expense of transmission range. Furthermore, the HC-12 supports a sleep mode, which would also be very useful for low-power applications when implemented. An example of a low-power application where these features would be useful is a solar powered radio-controlled car.



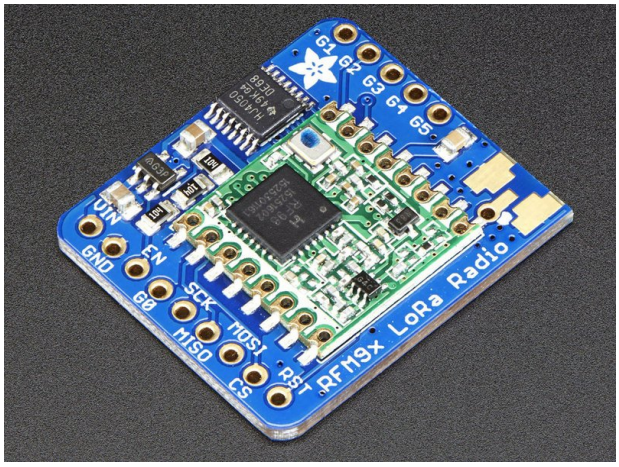
Example body of a solar-powered car

Additionally, the PCB hardware of the HC-12 module supports the connection of an external antenna. By default, the HC-12 is equipped with a spring antenna. However, other

antenna designs would be more suitable in various applications. For example, a directional Yagi antenna would be useful in extremely long range applications.

Commercial 433 MHz yagi antenna

There are also many other transceiver modules we could try for our RF-controlled robot car implementation. The ESP-8266 supports 2.4 GHz Wi-Fi communication. This would have a very high maximum bandwidth but significantly shorter range than the HC-12, which is useful if there were large amounts of data that needed to be quickly transmitted between the controller and the car, or if the system wanted connection to the internet. There is also the RFM95W LoRa Radio Module. This module supports extremely high range (up to 20 km) but less bandwidth. The RFM95W would be useful in applications where the remote and the car are very far apart.



RFM95W LoRa radio module

Conclusion

In this project, we explored the use of an HC-12 for wireless radio control of a car using a remote. We modified existing hardware to use the HC-12 module for wireless communication. We successfully implemented a reliable, low-latency one-way communication from the remote to the car for manual remote controls. Through this project, we explored and learned about designing communication protocols and the challenges behind constructing them to be robust and reliable.