Smart RC Car

CS122A: Fall 2017

Calvin Kwong

Table of Contents

Introduction **2**

Hardware **3**

Parts List 3

Block Diagram 4

Pinout (For each microcontroller/processor) 5

Software **6**

Implementation Reflection **7**

Milestone 7

Completed components 7

Incomplete components 7

Youtube Links **8**

Testing **8**

Known Bugs **9**

Resume/Curriculum Vitae (CV) Blurb **9**

Future work **9**

Extra Credit **9**

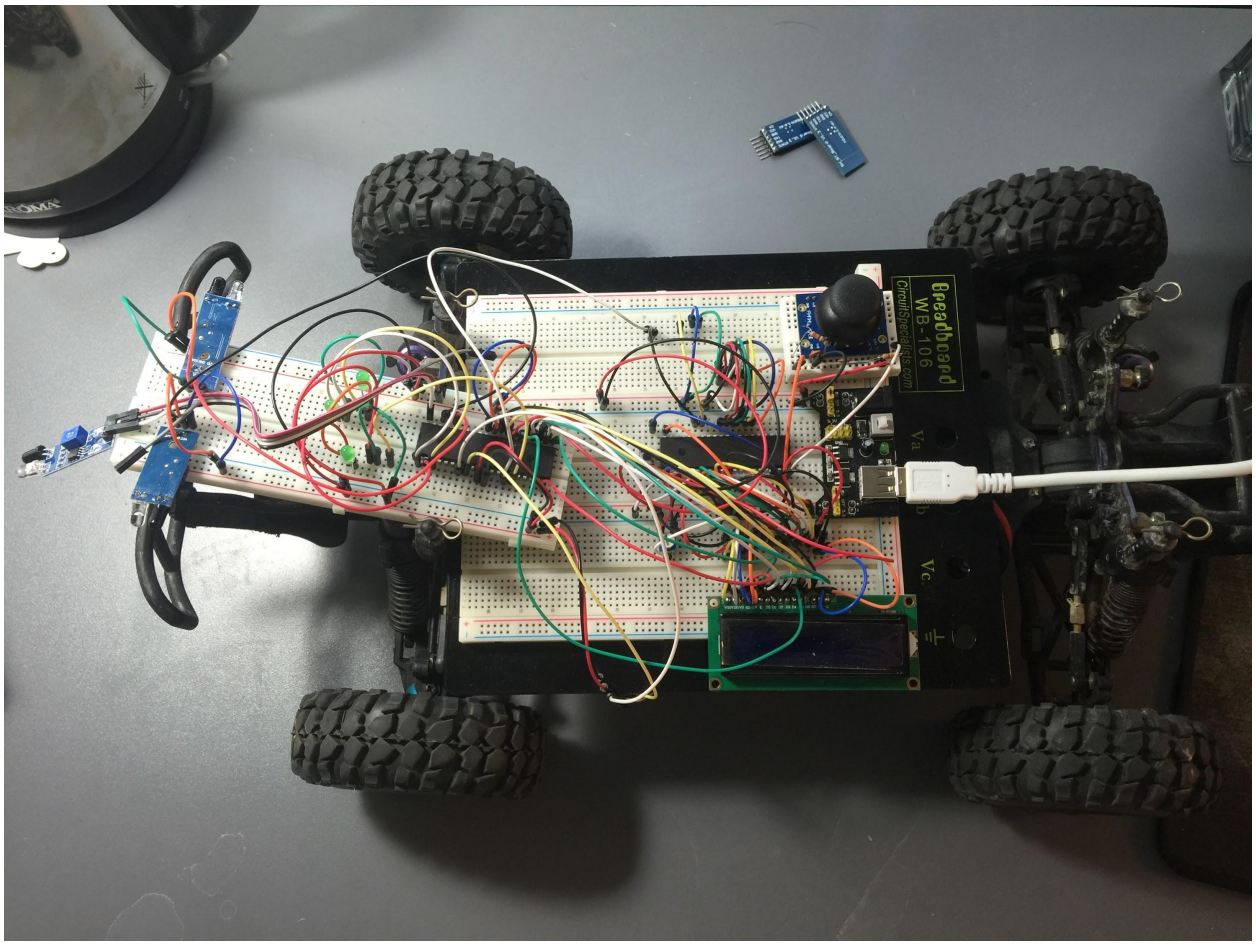
References **10**

Appendix **11**

Introduction

For this project I chose to create a remote controlled car with collision detection features. This project was inspired by electric skateboards as they both used similar design components. This project will be converting a RC (Remote Controlled) car, by removing the remote and receiver aspects and replacing them with microcontrollers. These microcontrollers will take the jobs of the receiver as a slave to the transmitting master remote.

The main method of communication was using bluetooth between the master and slave. The slave microcontroller handled the calculations to detect objects and corrected the steering and throttle to avoid the obstacle. The master microcontroller collected input from a joystick and broadcasted the information via bluetooth to the slave. However, in my case I had used USART instead of the planned Bluetooth communication.

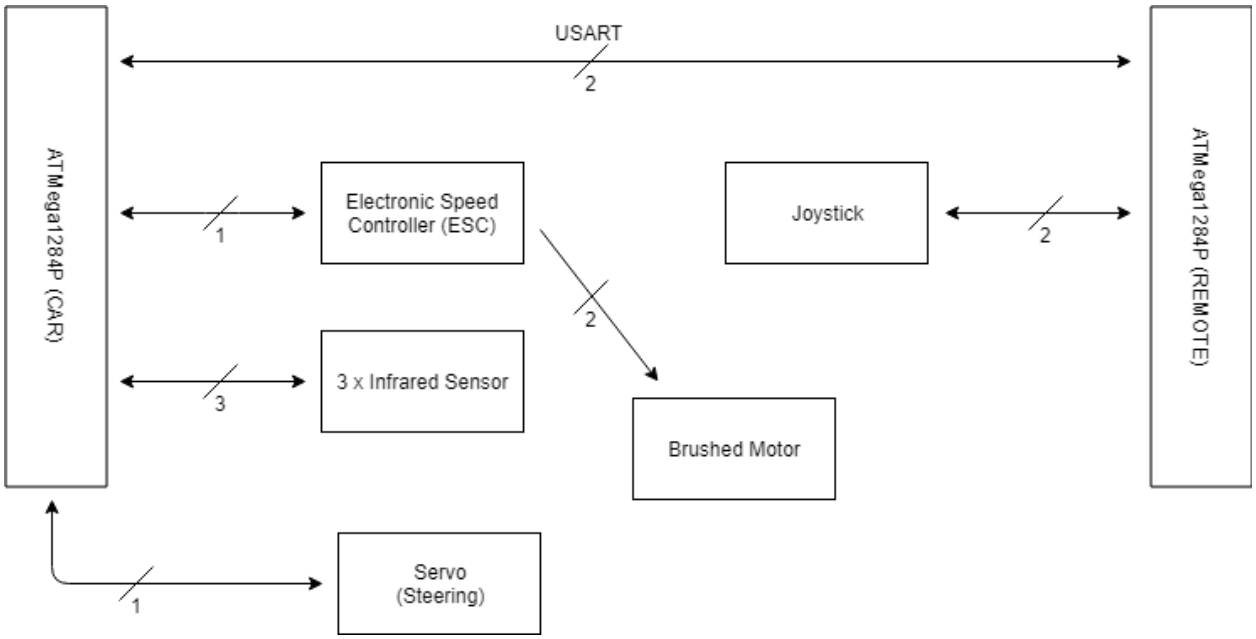


Hardware

Parts List

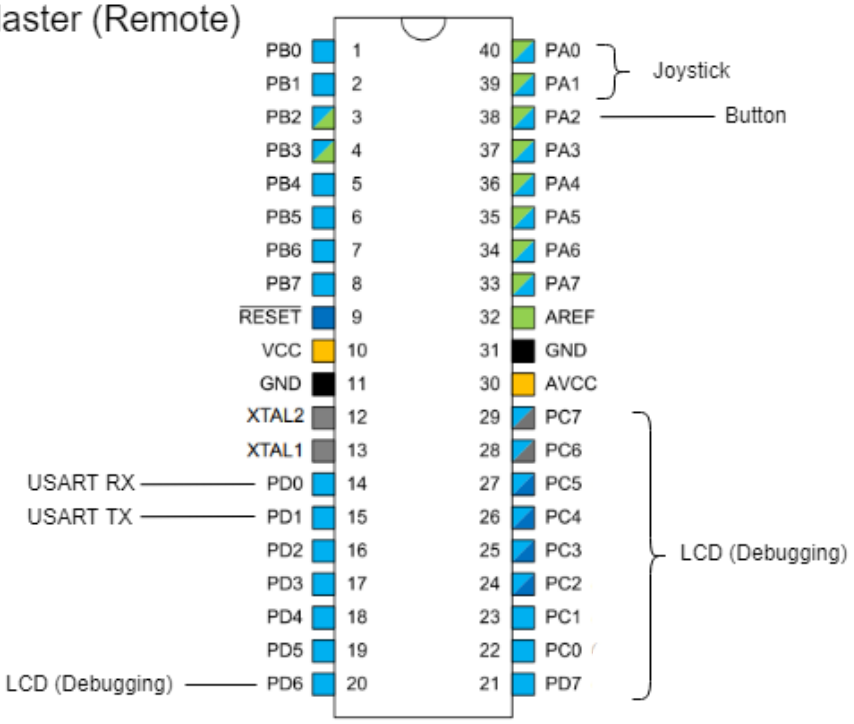
Part	Part #	Quantity	Price (optional)
ATMega1284P	ATMega1284P	2	
Joystick	Joystick	1	$5
LCD Screen (Debug)	LCD Screen	1	
RC Car	**RC-Car**	1	~$115
Electronic Speed Controller (ESC)	**Electronic Speed Controller (ESC)**	1	
7.2V Battery	7.2V Battery	1	
Steering Servo	**Steering Servo**	1	
Chassis	Chassis	1	
Bluetooth Modules	**HC-05**	2	$(2 x 10)
Infrared Sensors	**Infrared Sensors**	3	$(3 x 1) (10 Pack)
		Total	$150

Block Diagram

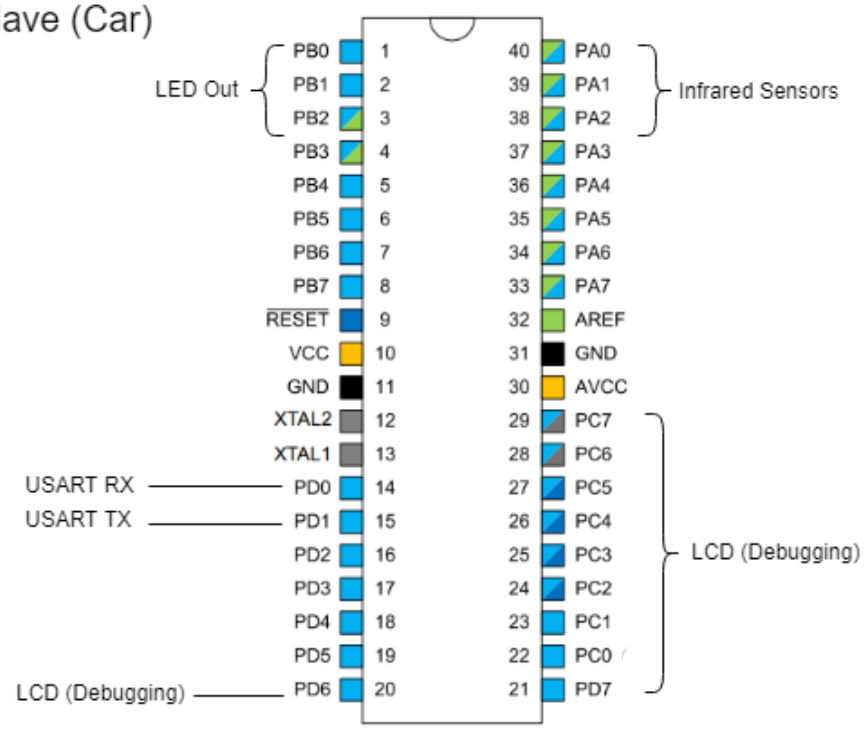


Pinout (For each microcontroller/processor)

RC Master (Remote)



RC Slave (Car)



Software

The two tasks communicate using two wires via USART. The remote or master task sets the highest bit depending on what value is transmitted. The control or slave task then can differentiate between the throttle and steering by looking at the highest bit.

The task for the remote(Master) receives input from the joystick for the steering and throttle. It then converts the values from the ADC to a percentage value in 8 bits to transmit to the microcontroller on the car using USART.

The task for the car(Slave) receives the transmitted information from the remote via USART and uses those values to set the throttle and steering angle. In addition to controlling the steering servo and throttle, this task also handles the collision detection. If there is an object in the front or in the side directions, it will adjust each value accordingly to avoid a collision.

Implementation Reflection

Overall, this project felt like an accomplishment and I am proud of it. Before I started this project, I had no knowledge of how the internal components of an RC car functioned and how they interacted with each other. Most of my time for this project was dedicated to researching parts and trying to figure out how to interface the microcontroller with it. In the world of RC cars, not much information is given on how to interface the internal parts as the whole industry is oversimplified for the hobbyist. It was pretty difficult finding the right information for both the RC internal components and how to program the hardware features of the microcontroller.

If I had another opportunity to redo this project, I would have invested in purchasing a RC car instead of borrowing one from a friend and bought more spare parts. The investment for one of these cars is about $150 dollars, which was too high for me at the time. While borrowing this RC car, it was difficult to test because I was worried about damaging any of the components. While debugging my project, I also ended up frying a lot of my limited parts. I enjoyed working with RC toys as my dad used to have a few models. Testing the parts of the RC car and understanding how they worked was also a pretty rewarding experience.

Milestone

My milestone was to control the RC car using just the ATMega1284 by the second week of the project. I did not meet this milestone because I had some difficulty finding and researching an affordable RC car to purchase. By the time I was able to get a RC car to test, it was near the end of the third week for the project.

Completed components

For my project, I reached the 80 - 90 range. I was able to control the steering servos using the microcontrollers through USART and implemented a collision detection system. I was unable to reach the 90-100 portion of my project because I had some hardware issues. During debugging I discovered that my ESC increased the overall circuit for the RC car's electrical system to about 6-7.2 volts and fried my bluetooth board.

Incomplete components

In my proposal I planned on having a drivable RC car with proper collision detection by the demo date. While working on the project, I unfortunately ran into some issues that prevented me from completing my project. In the earlier section I had mentioned that the ESC increased the overall voltage for the microcontroller circuit along with the bluetooth and servo. This voltage increase, from 5 volts to 7 volts, caused the servo and bluetooth module to overheat. Unfortunately my bluetooth module was fried and I was unable to reorder another by the time of

the demonstration. The issue with the voltage forced me to disconnect the ESC and drive system in order to prevent the servo from being damaged.

Youtube Links

- Short video: https://youtu.be/RYqLwpa4hIM
- Longer video: https://youtu.be/zplhUXiXwTA

Testing

My first test case for this project was to see if the remote microcontroller and the controlling microcontroller received proper values from each other. The controlling microcontroller would then convert the values into the PWM range used to drive the steering. For this test case, me and my friend from CS 120B had tested it. While testing this I noticed that the steering servo was unable to turn fully. I had isolated this bug and corrected the PWM ranges to run the servo.

The second test for my project was adding on the infrared sensors for collision detection. For this test, I placed objects around the RC car and tested if the sensors would flag the object. If an object was flagged the microcontroller would prevent the car from heading into that direction. For this test case, I had my partner from CS 122A test this portion with me. I had him controlling the joystick while I had placed objects around the car. The steering or throttle should be limited if an object was sensed. The results of the test was that it worked correctly, however I found out the range for the infrared sensors were very short.

Infrared Sensors:

For this component, I tested it by placing objects in front of the sensor and seeing if the microcontroller receives a flag. If a flag was received, a LED would light up.

Joystick:

For this component, I had used previous code to get the ADC values for the left, right, up and down values.

Bluetooth Modules:

For these components, I had tested them by pairing and had them try to send values to each other. I had tested them by running a lab we did on USART and seeing if they received the input.

Steering Servo and Electronic Speed Controller (ESC):

For this part, I had written a program that adjusts the PWM every second that would increase the width of the PWM. I had an LCD display that would show the width in milliseconds.

Known Bugs

Prior to frying my Bluetooth modules, I had a running example for it. I followed a guide on how to set up the modules and paired it correctly. While transferring data I would not receive the correct information, I believe this was an issue with the baud rate or I had received faulty parts. I was unable to debug any further as the modules stopped responding to my programming commands after being fried. If I were to continue my project, I would most likely order replacement parts and continue further testing.

I would not consider my ESC issue a bug as it is more related to a hardware issue. In order to correct this issue I would need to add a battery eliminator circuit (BEC), which essentially lowers the battery voltage to one usable by the bluetooth and microcontroller.

Resume/Curriculum Vitae (CV) Blurb

I created and designed a Smart RC car using two ATMega1284P. This project uses two microcontrollers, one to control the car and one acting as a remote. The remote communicates with the car's microcontroller using USART. The car's microcontroller can then adjust the throttle to the ESC (Electronic Speed Controller) and the steering servo using the hardware based timer PWM. They were both designed using state machines and scheduled using FreeRTOS.

Future work

If I were to continue this project, I would most likely fix the errors that I had and implemented it using additional parts. The issues with the ESC and bluetooth communication could be fixed by using a Battery Eliminator Circuit (BEC) to convert the higher battery voltage to 5v. Instead of using the infrared sensors, I would have used the sonar sensors as they have a longer range. A custom PCB could be printed to minimize the footprint on the car. The remote microcontroller could be shrunk down into a PCB and 3D printed in the shape of a remote. My idea for a Smart RC car is not the most genuine, but I would still bring it in for an interview if I applied to a company related to embedded systems.

Extra Credit

Include a link to a public DIY for your project: <link>

References

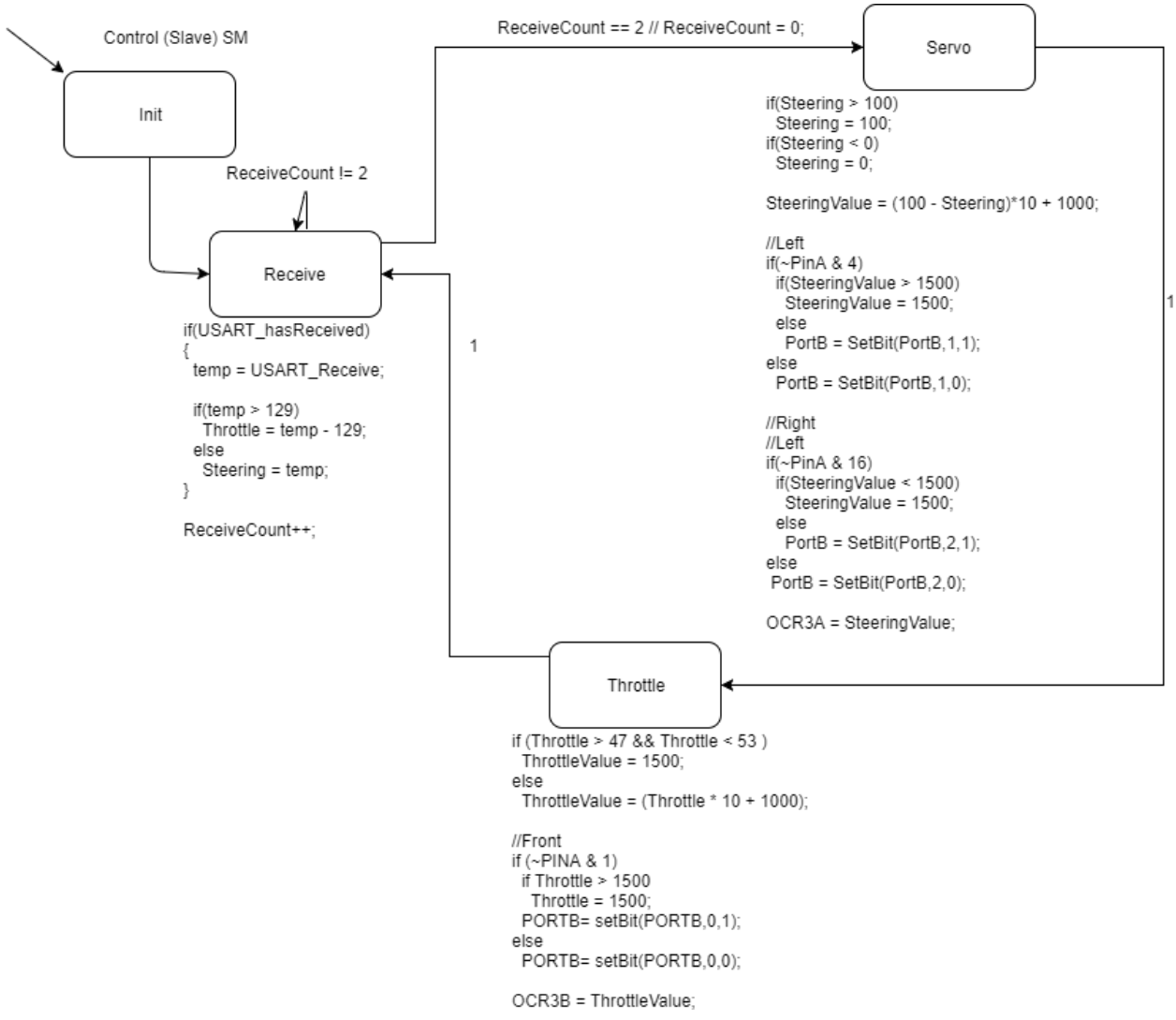
My main inspiration for this project was the electric skateboard, ZBoard. However, the process of designing and building an electric skateboard would take longer than what this project

allowed. Building an electric skateboard would use similar parts such as the ESC, Bluetooth and microcontrollers. Machining the wheels to allow it to interact with the motors would require experience with CAD software and finding the equipment to do so.

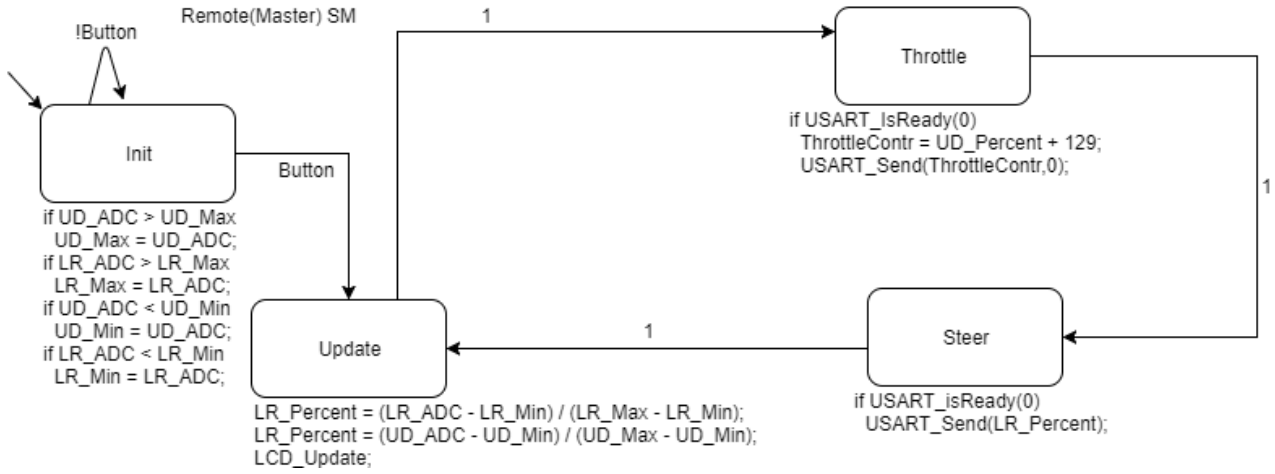
Reference Material:
-For Bluetooth Pairing
http://www.instructables.com/id/AT-command-mode-of-HC-05-Bluetooth-module/
-ATMega1284 Manual

Appendix

Period: 50ms



Period: 50ms


Remote(Master) SM
!Button
Init
Button
if UD_ADC > UD_Max
UD_Max = UD_ADC;
if LR_ADC > LR_Max
LR_Max = LR_ADC;
if UD_ADC < UD_Min
UD_Min = UD_ADC;
if LR_ADC < LR_Min
LR_Min = LR_ADC;
1
Throttle
if USART_IsReady(0)
ThrottleContr = UD_Percent + 129;
USART_Send(ThrottleContr,0);
1
Steer
if USART_isReady(0)
USART_Send(LR_Percent);
1
Update
LR_Percent = (LR_ADC - LR_Min) / (LR_Max - LR_Min);
LR_Percent = (UD_ADC - UD_Min) / (UD_Max - UD_Min);
LCD_Update;