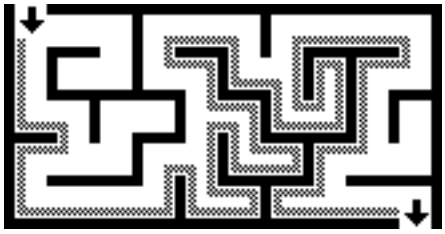
In Final Project, we combine some application we learned in this class on BBcar.

(1) Implementation and Algorithm

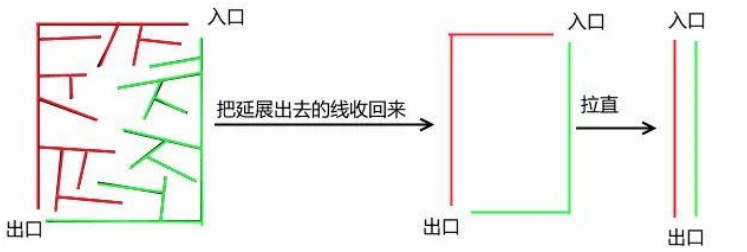
Basic map navigation:

Main algorithm

Considering route as a maze, which has an entrance and exit at the outer boundary. If we traverse the map with wall contact to right hand (BBcar with right wheel). Then it will reach the end or go back to entrance eventually. The algorithm is called “wall follower” or so called “right-hand rule.”

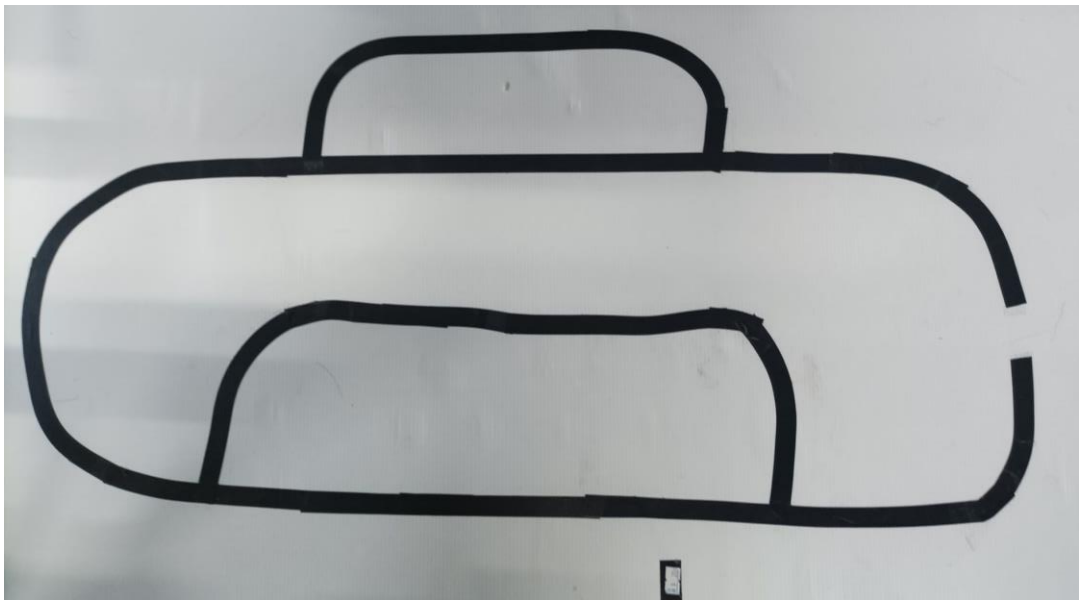


Since if we try to pull straight the boundary of maze. We can find the exit.



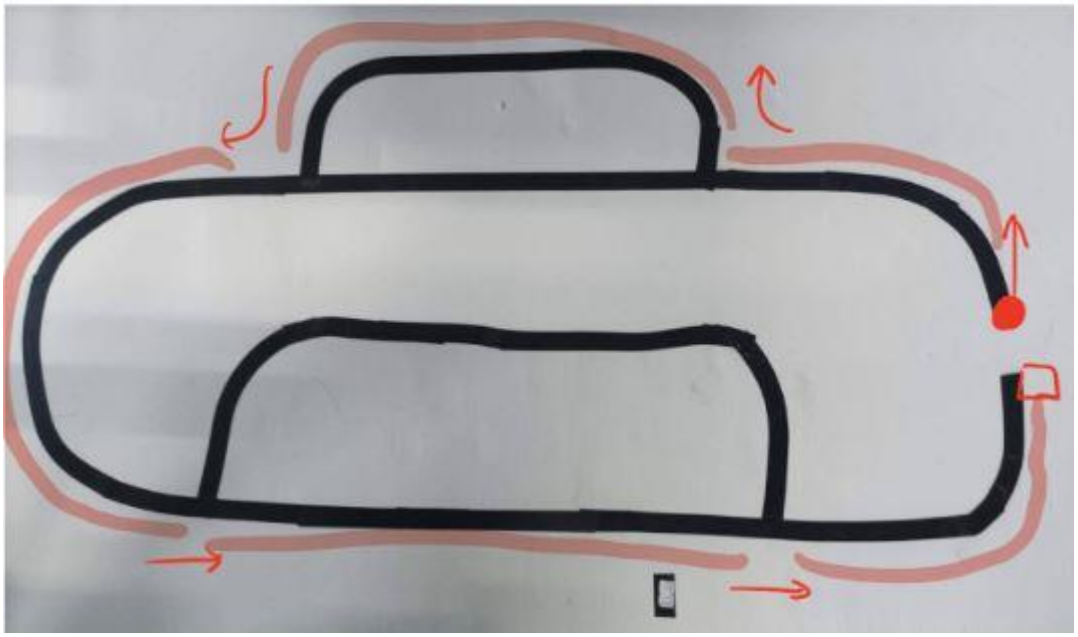
So by using this algorithm, BBcar supposes to find out its way to the end of route.

Map design:



Here’s the map I used in DEMO section. And here’s how the car suppose to

traverse. If we apply the algorithm.



To follow the route, we use QTI sensor that detect different kind of situation.

Pattern	Bits code	Type	Right wheel	Left wheel
	0b 1000	Big left	Forward	Backward
	0b 1100	Medium left	Forward	Stop
	0b 0100	Small left	Forward/Big	Forward/Small
	0b 0110	Straight	Forward	Forward
	0b 0010	Small right	Forward/Small	Forward/Big
	0b 0011	Medium right	Stop	Forward
	0b 0001	Big right	Backward	Forward

Other setting in discussion part.

Marker and task identification

Now considering the cross section that we may confront when we traverse the map. Using QTI sensor to detect corresponding mark and make decision.

	Pattern 1	Pattern 2	Pattern 3
Picture			
Type	Right turn	Cross section	Left turn
Bitcode	0b0111	0b1111	0b1110
QTI			

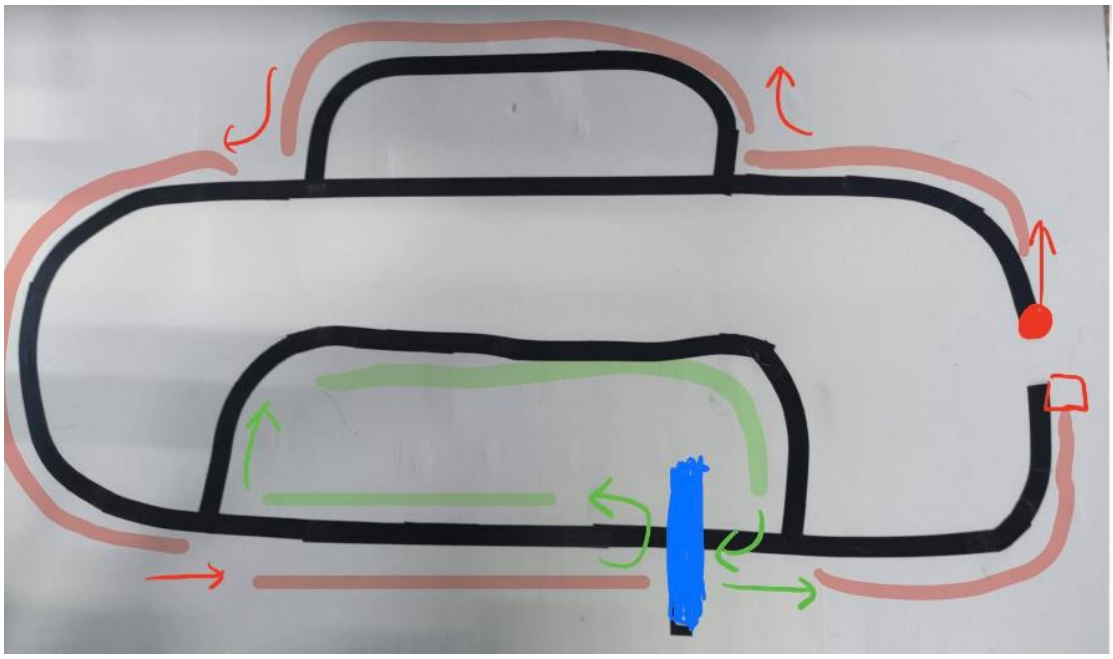
Application	By algorithm, BBcar need to turn left. It will turn 90 degree to left and resume following line.	By algorithm, BBcar will turn right. But there are some cases that it needs to turn left, which alter some of the algorithm with Ping sensor.	By algorithm, BBcar will neglect the pattern and keep following line.

Also, for start and end point there is a blank space that QTI sensor will read 0b0000, which indicate it reach its destination and will stop moving.

Ping Sensor

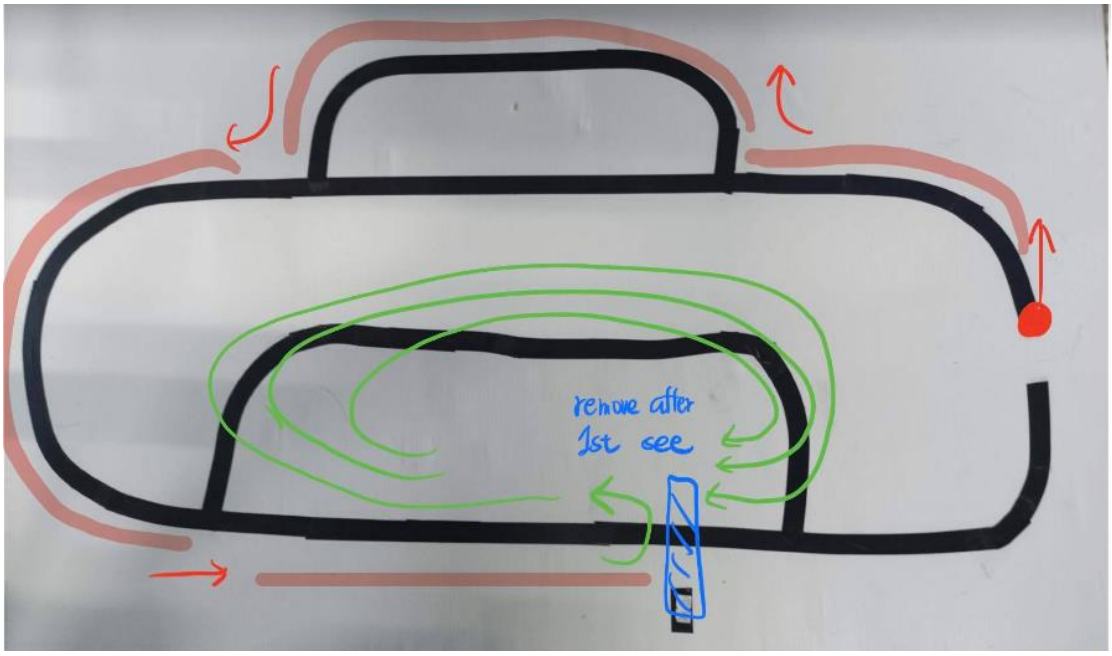
With ping sensor, BBcar will detect whether there is object in front of it every 500ms. If it detected there is some obstacle on its route. It will move backward with a short distance, then turn 180 degree and resume following line.

However, there are some problem when we apply with our algorithm. Consider the map which obstacle mark with blue color and route with red color and green color.



When the car sees the obstacle, it will turn 180 degrees, and it will turn right at the cross section and sees the obstacle again, and turn 180 degrees then reach the destination.

However, if we remove the obstacle when the car first time see it.



The car will go into an infinity loop that it won't ever came out. Since we didn't follow the algorithm which leads BBcar's right wheel go into inner wall, where we need it to maintain contact to the boundary of the route.

So to do with this problem, when BBcar detects obstacle on its route, it will turn left when it sees the cross section (T shape or ┼ shape) to avoid go into infinity loops. (TA asks question related to this)

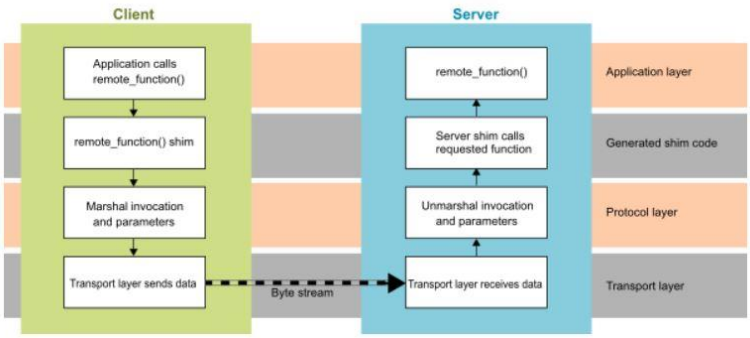
By doing this, BBcar can detect any obstacle on the route and try to apply another branch when original route is block with obstacle.

Xbee and eRPC implementation

To apply this implementation, we take reference from its github.

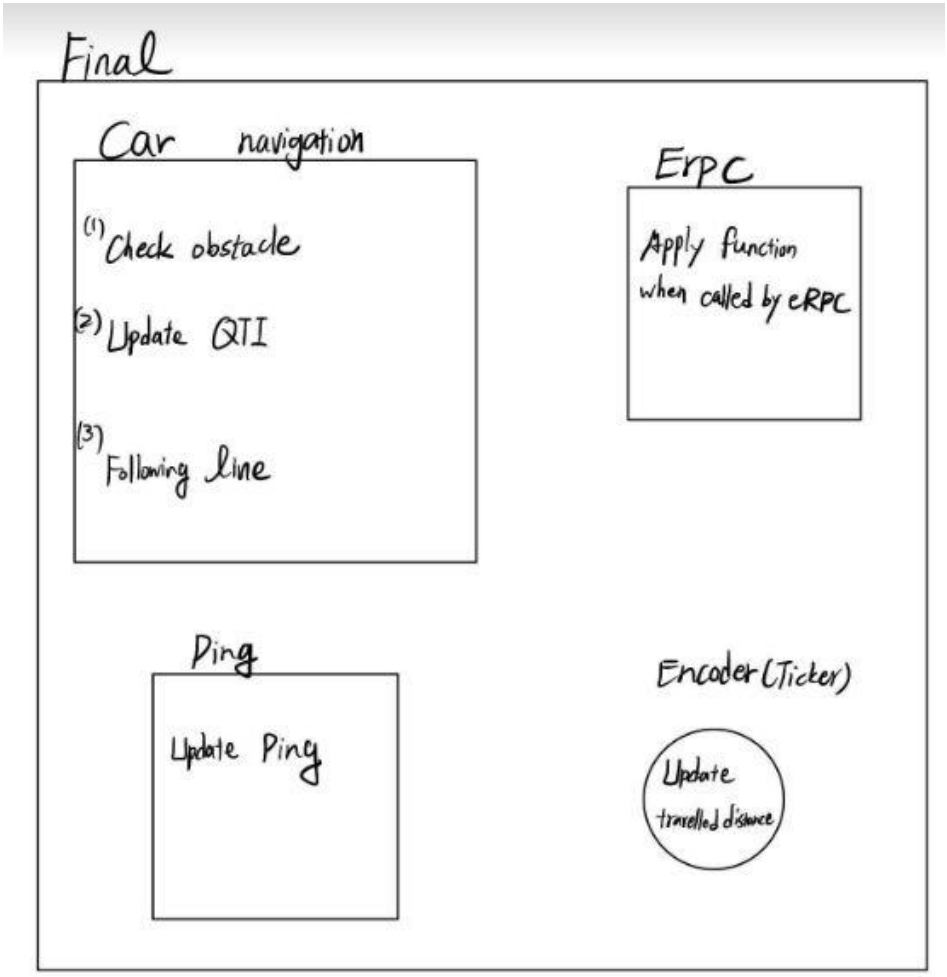
When a remote function is called by the client:

- The function's parameters and an *identifier* (for the called routine) are serialized into a stream of bytes.
- This byte stream is transported to the server through a communications channel (IPC, TPC/IP, UART, and so on).
- The server unserializes the parameters, determines which function was invoked, and calls it.
- If the function returns a value, then the value is serialized and sent back to the client over the communications channel.



When the Client(Python) called function, the Server(Mbed) will implement the function and return value through UART(XBEE), Functions that return value can catch the message from the mbed. So I create a function that insert one parameter to decide which data I want to get. One is the distance that BBcar has traveled, which is the data recorded by the encoder. Another is the distance in front of the car which is measured by the Ping module.

Combination



Here's the final block diagram, combine them together with thread and we are ready to go.

(2) Validation and Result

The car navigation is shown in demo section on 6/1

And eRPC is shown below

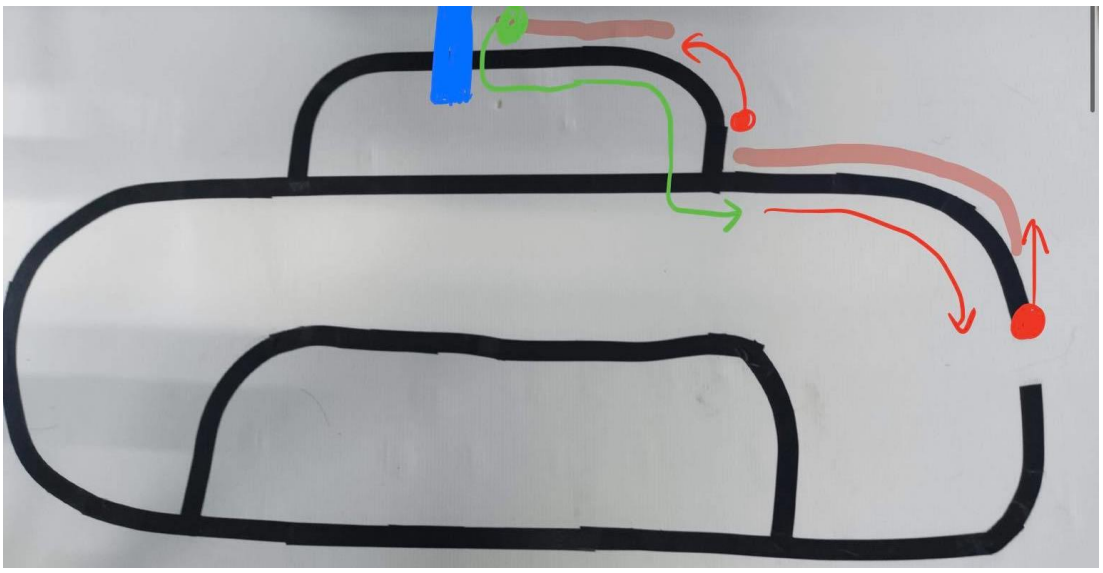
```
HELLO, welcome to erpc service

Encoder data insert 1, Ping data insert 2: 1
Traveling distance:    0.0  cm
Encoder data insert 1, Ping data insert 2: 1
Traveling distance:    60.592185974121094  cm
Encoder data insert 1, Ping data insert 2: 2
Ping with distance:    149.44447326660156  cm
Encoder data insert 1, Ping data insert 2:
```

(3) Encounter issue

(a) The algorithm

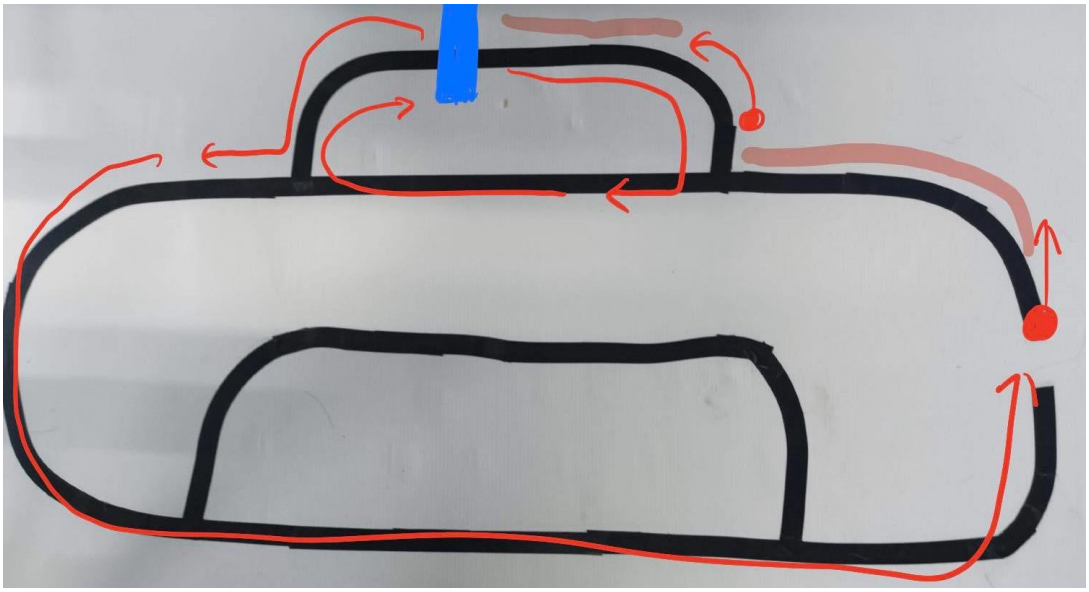
Though the wall follower algorithm seems powerful, there is still defect in this modified algorithm, which will turn right one time at the cross section after it detected obstacle and reversed.

Consider the map below.



If we apply the modified algorithm, it will go back to start point again.

However, the original algorithm should be applied by following picture:



Which always turn right when it sees the cross section and the obstacle is always there.
So if the requirement is restricted to that obstacle won't disappear and not considering the volume of BBcar, it will be easier.

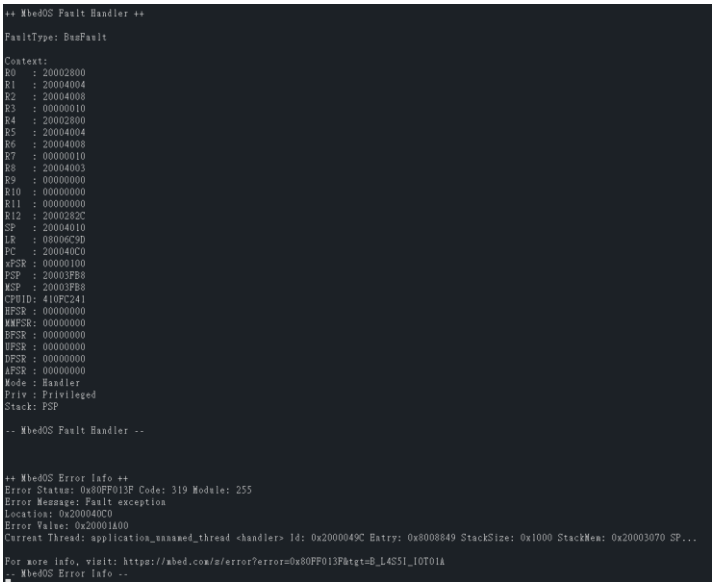
(b) The obstacle

Please place obstacles on the map and use LaserPING (PING) to detect obstacles dynamically and navigate to avoid the obstacles. For example, if there is no branch between BB Car and an obstacle, BB Car will go in reverse to the last branch.

In this requirement I misunderstand it. I think that obstacle will randomly appear in the route and may disappear. So if the marker only work for indicate turn right or left at the next cross section may not work if the obstacle appears randomly. Considering this problem, I choose wall follower algorithm which will suite any kind of cases even with different map it will still work.
Unfortunately, when I saw other's demo where their obstacles are fixed, I know I take this quest too seriously and ask myself for trouble QQ.

(c) ERPC thread and volatile declaration

In forum in eeclass, some people encountered that mbed crash problem.



So do I, and I try to find out bug for almost 5 hours.

And there are some candidates

(1) Python asked for data too fast

Sine data is transmitted with UART and eRPC function is put inside a thread, if user call function rapidly and mbed is not able to respond to its requirement, it may crash.

(2) Volatile declaration

Since Ping module and encoder is not built-in module on Mbed, some variables need to be declared by volatile, also some variables are transmitted through XBEE, I also declared them with volatile to avoid problem.

(3) Infinity loop

In car_navigation thread, I have an infinity loop with empty assignment and task like this:

while(condition) ;

Also, in BBcar library, there is also a while loop that may lead to crash, I found out this problem when not applying PING simultaneously. Also I make it refresh rate lower to 1 time / 0.5 second.

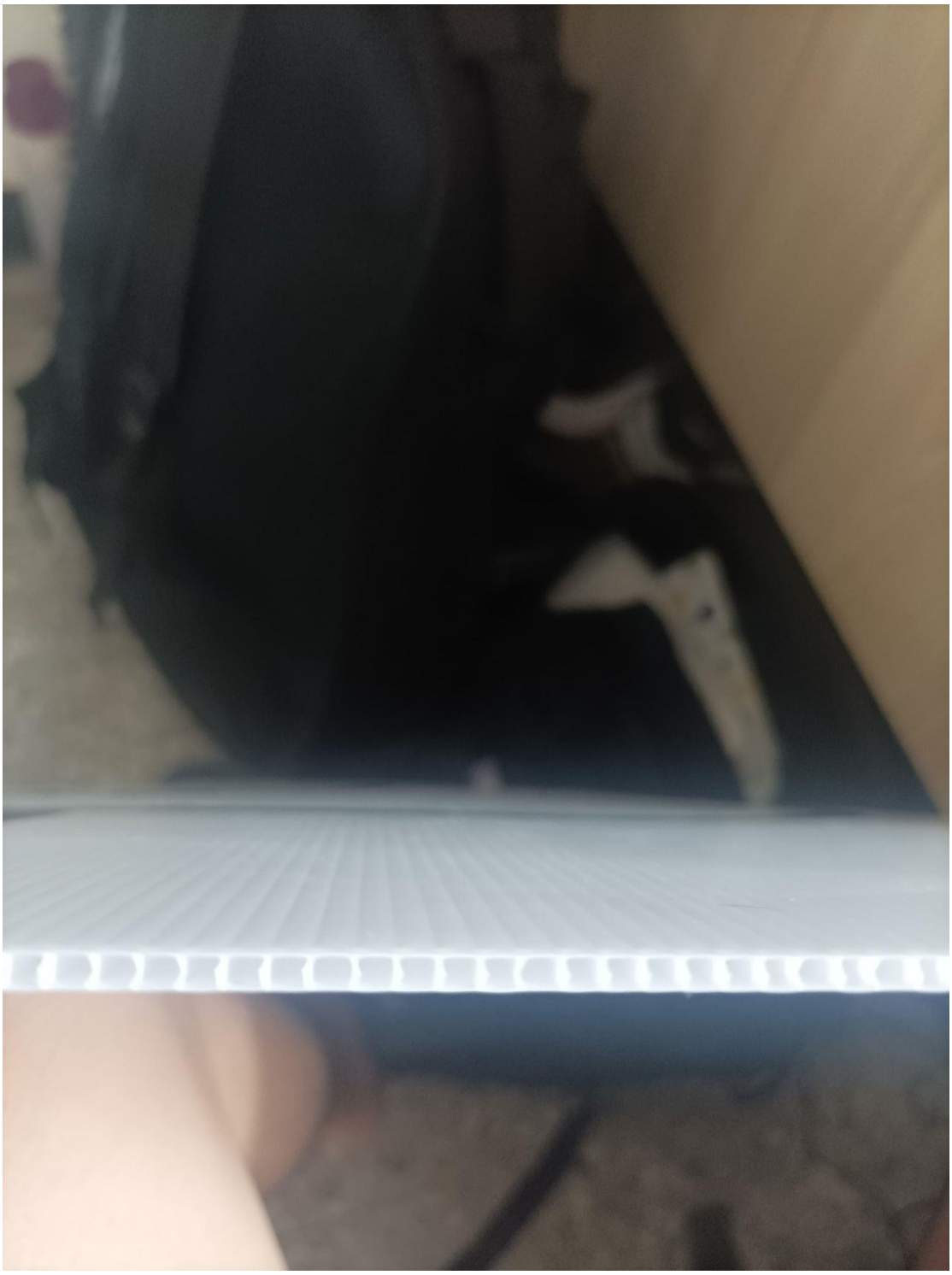
(d) Map material and texture

When designing map, the space in dorm ren(仁齋) is crowded,

Moreover, it has some hole on the floor that will affect QTI sensor.

So at first, I bought a big white paper, while I encountered some problem that the paper is not so flat. There are some bulges which also affect QTI sensor.

To avoid this, I bought another plastic plate, but there are still some problems, there are straight lines with dents and bulges as shown below.



Which also leads to QTI sensor problem, in demo section, the car should recognize the cross section, however because of the material of plastic board, it took it as a right turn.

I figure out that it will only recognize the pattern with line and line has about 75 degrees, not vertical between them. While in demo section, the tape was not stick to the plastic plate properly so I paste it again. And the angle was not correctly adjusted.

Hope that I can have a bigger room to test the car. My roommates almost hit the car when they opened the door. How sadly it is QQ.

(4) Discussion

I spent a lot of time on final project. And eventually found out that I took the question too seriously. Considering the obstacle will appear randomly then the difficulty of task raises rapidly, since we need to consider more problem that marker and original route when encounter branch-changing problem.
When I saw others' demo section. Then I found out that the obstacle is fixed. Everything go easier, I just need markers that decided turn right or left and cross section with T or ┼ shape, then everything is under controlled. Don't need to bother with dynamically appear obstacles.

Also the thread crash problem also cost me a lot of time. Need to find out a way that Mbed transmits data back to Python through eRPC function. Moreover, need to find out what's going on when encounter Mbed crash with threads, it looks simple and easy to implement every task independently. While to combine them is another big task. Can't use polling in final project since erpc function will block everything. I can say that we concentrate everything we learned from this class to implement BBcar.

Also it's quite hard to adjust the car itself, the wheel needs a lot of try and error to find out the best setting to decide the magnitude of speed when turning. I also change setting in BBcar library with higher refresh rate to wheels.

I think I learned a lot from final project and this class. In microprocessor lesson we learned some basic knowledge. And in this lab we learned more, like oscilloscope, thread, event queue, basic ML, MQTT, TCP/IP, XBEE those are I didn't know before. And some of them is quite powerful and useful. Thanks to professor and TAs, I really learned a lot.