Inha University in Tashkent
21.05.2019

SMART CAR
Documentation
Code Poltergeists

ICE - U1510047 - Voriskhon Ganikhujaev
ICE - U1510013 - Bobur Zaitov
CSE - U1510362 - Akbar Karshiev
CSE – U1510391 – Khikmet Bakhadirov

Inha University in Tashkent
School of Informational and Communicational Engineering
(SOCIE 15-2)

Submitted to Hakil Kim, Jangwoo Kwon, Dr.
as a Capstone Design assignment (ICE4020)

TABLE OF CONTENTS

1. Main goal ……..……..……..…… 3

2. Lane detection part of smart car ……..……. 3

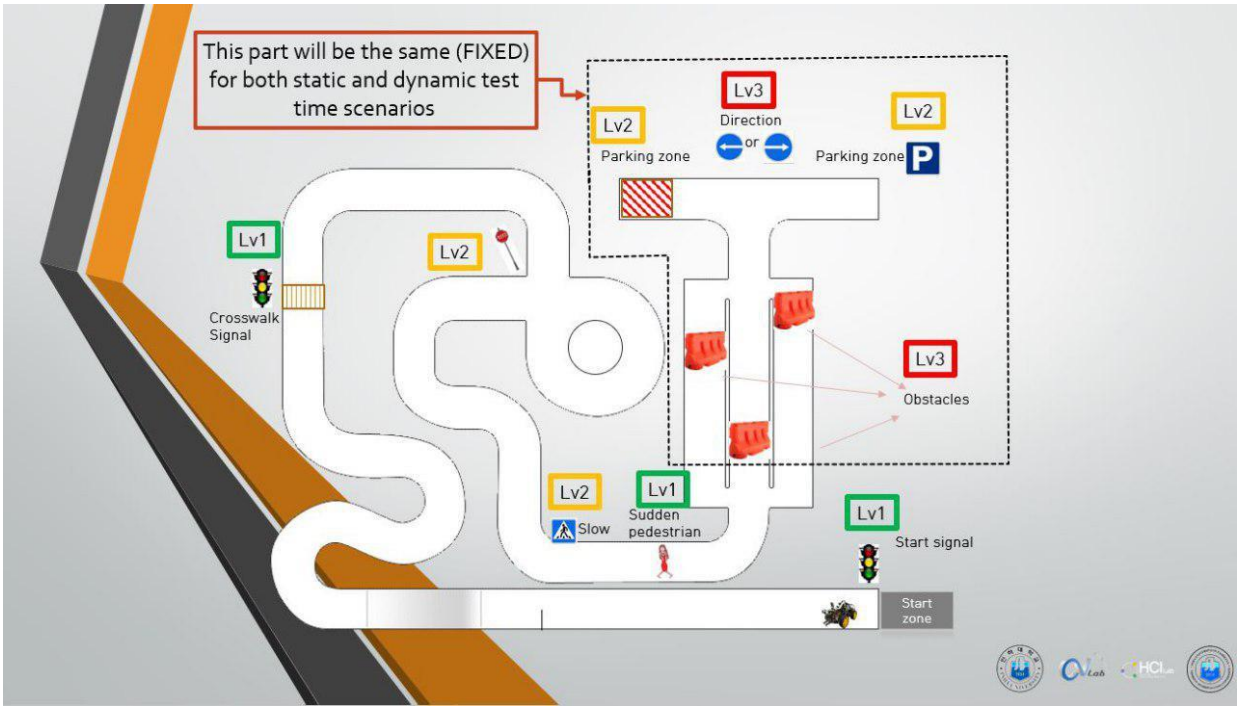
3. Sign detection based on shape and color …….. 4

4. Gathering all parts ...……...…… 6

5. Summery ……..……. 9

1. Main goal

Develop a smart car on the base of tasks such as motor control, IR and Ultrasonic sensor management (black line tracing, obstacle detection), image processing (traffic color recognition and movement), movement detection according to detected lines, sign detection. All above mentioned issues should be solved in the team by professionalism, roles and given parts to each other.

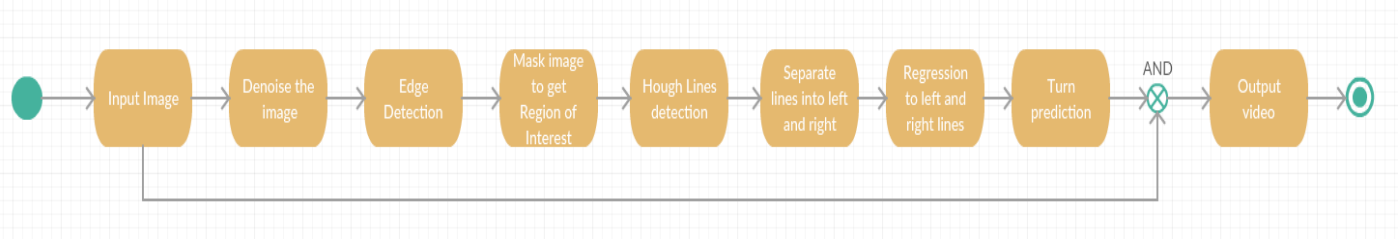


2. Lane detection part of smart car

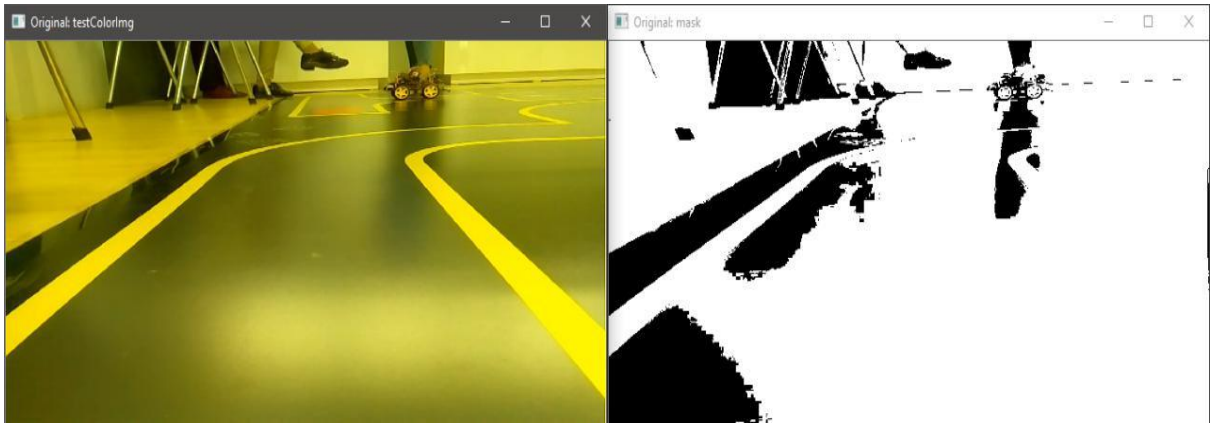
Our Lane Detection algorithm was based on information from source:

https://github.com/MichiMaestre/Lane-Detection-for-Autonomous-Cars

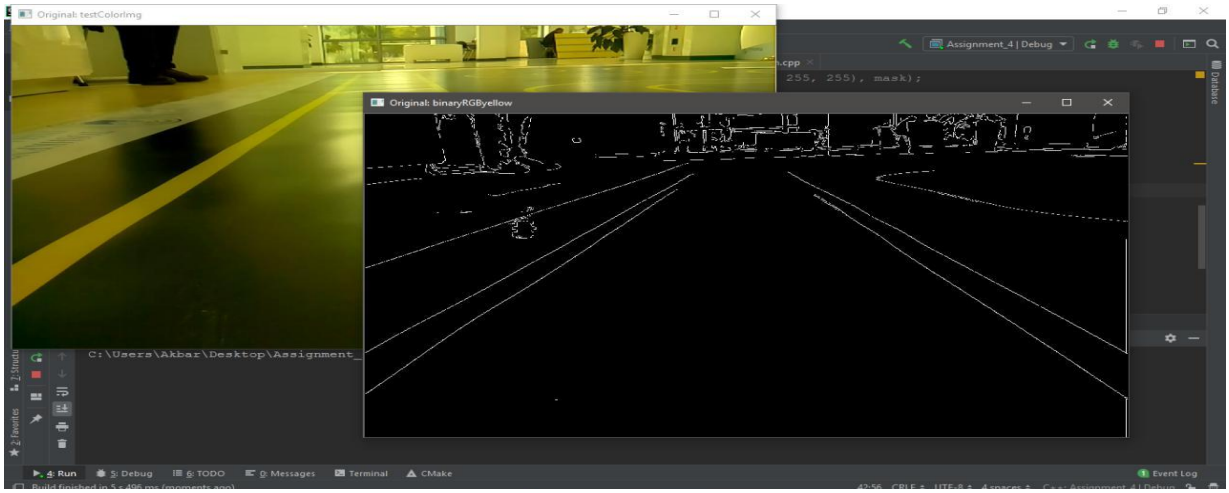
At the beginning of the development of Lane Detection program for smart car we were going to use standard algorithm of Lane Detection using color space separation method, where yellow and white colors of image converted to HSV color space, were used to find yellow and white line on the map.



However, because of reflection on the glossy map and inappropriate lighting of environment we were unable to use this method. Here is the proof:



As a result, we changed half of the code from the source above to fit our requirements. Then, we used Canny Edge detection method directly onto denoised input image:



As it is shown in the flow chart above after getting edges from image we used right and left lines separation method where according to angles of slopes and relative position of lines to the center of the image they are separated.

Then, the least squares method of linear regression used to these separated right and left lanes to get edges of Lanes.

For turn prediction step, we set rotation speed of left and right motors according to vanishing which we find using detected lanes from previous step.

3. Sign detection based on shape and color

We decided to use basic approach to differentiate our traffic signs.

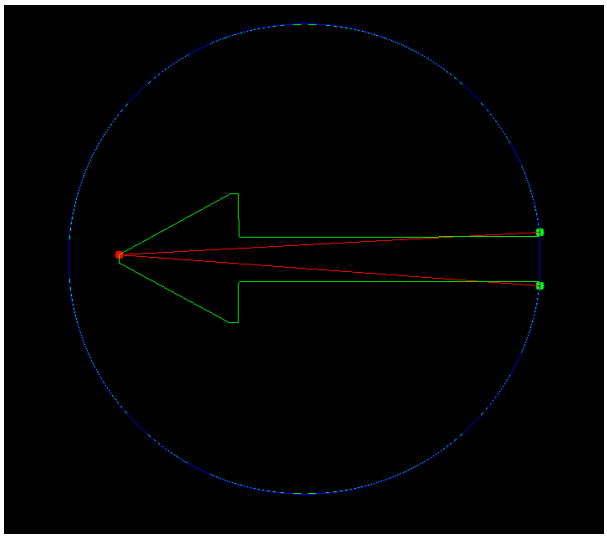
Pic-A

Lets see “main.cpp” file first. In main part you can use image or camera for running our code.

```
bool isImage = true; // if you diceded to use image; otherwise ==> false
```

Then connection to image or video camera and detect either it available or not. Soon it takes frame of image/video and provides it to “findShape(Mat src)” function (where src is our frame).

There it calls “getImageChannel” function in “Colors” class and provides frame and color name (blue or red). Soon this function returns masked frame depend on which color was provided. Then we provides masked frame, masked color name (as string) and thresh (150) to “getShapes” function in “ShapeDetector” class. Let see closer to the operations inside it. In this function we started with converting our masked frame to gray, then blur it and applies threshold. Soon we finds the contours, looping till the end of the contours and use in the loop approxPolyDP to find vertices of the contour. Skip in small contours and then skip non convex shape (except blue color). It is easy to find caution (red triangle) by color and vertices number. If there is more then 3 vertices we should find corners degree of the contour. If our contour rectangle and it has inside contour (by hierarchy[i][2] != -1) – it is parking sign or pedestrian. To find it out we should see next contour (which will be inner ones) and if we found triangle –



Pic-B

“pedestrian”, else “parking”. The stop sign has tricky park, because it finds out from 8 to 11 vertices depends on the turn of this sign. So we take this intervals to make it easier to detect. “Circle” sign (Circle-ahead in our case) has approximately 8 vertices and color blue. If our contour is not entered in if statement:

```
if (( approx.size() >= 4 && approx.size() <= 8)||(approx.size() >= 8 && approx.size() <=
11 && color == "red"))
```

So it is circle. We find out the center of this contour and checking to nonconvexity of this shape (due to non-solidity of the shape). We start to find hull with help of convexHull to identify convexityDefects of the contour. Filtring it by depth (see ‘Source1.cpp’ in ‘shape/additional_codes’ file.) ‘Pic-B’. In ‘Pic-B’ demonstrated left sign with depth marked as red dot. We should find the depth and find out the delta of center point and depth. If delta y more than delta x → up or down sign, else left or right. We are not interested in up and down, so we are using left or right and name our contour respectacly.

In the pictures below, you can see the output in real case:

4. Gathering all parts

For interaction with real world Smart Car has Bottom and Top IR sensors, Ultra Sonic sensor and Camera. Bottom IR must detect road lines and hold car on the road, Ultra Sonic must detect obstacles in front of the car, Top IR must detect obstacles from left and right sides that Ultra Sonic cannot detect, Camera must detect crosswalk, traffic light, traffic line and traffic signs. All these features must work properly and at the same time during the executing process. For this we

used multithreading, it easier to implement rather than multiprocessing. All features were arranged in threads with custom privilege levels and send command to Motor control functions. Only Top IR and Ultra Sonic was placed in a single thread because their function is to detect obstacles and for better work of other functions that need their own threads. Motor control functions were placed in main thread and has no privilege level because it must only receive commands from other threads and execute them.

privilege_level and *command* are global variables, main thread read command variable and in if else statement choose which motor control function to call

```
/// Global Flags for motor control
static int privilege_level = 0;
static int command = NOTHING; // command to imlement
```

Privilege levels of threads

```
// privilege levels
#define SHAPE_DETECTOR_PRIVILEGE_LEVEL  5
#define ULTRASONIC_PRIVILEGE_LEVEL      4
#define BOTTOM_IR_PRIVILEGE_LEVEL       3
#define TOP_IR_PRIVILEGE_LEVEL          2
#define LINE_DETECTION_PRIVILEGE_LEVEL  1
```

Commands that functions send

```
#define STOP                0
#define FORWARD             1
#define LEFT                2 // go forward and turn left
#define LEFT_POINT          3 // stay and turn left
#define RIGHT               4
#define RIGHT_POINT         5
#define SPEED_DOWN          6
#define SPEED_UP            7
#define STOP_DELAY          8
#define LEFT_CURVE_TURN_20_60       9
#define LEFT_CURVE_TURN_20_75       11
#define BACK                12
#define NOTHING             -1 // give way to other commands
```

SetCommand() is the function To control sequence of command execution. Any Thread can call SetCommand function and send it`s privilege level and command to execute. SetCommand function will change global variables if received parameters satisfy if else statement.

```
void setCommand(int lvl, int com) {
// if current command is NOTHING then it does not matter which command to do next
    if(command == NOTHING) {
        privilege_level = lvl;
```

```
        command = com;
    } else {
// if current command is not NOTHING then it does matter which sensor can execute
command
        if(lvl >= privilege_level) {
            privilege_level = lvl
            command = com;
        }
    }
}
```

Bottom IR

Bottom IR needed to detect road lines. Send 1 if line detected and 0 otherwise. If left IR sends 1, Bottom IR thread will send RIGHT_POINT command with BOTTOM_IR_PRIVILEGE_LEVEL 3 to turn right and LEFT_POINT otherwise. There are 2 lines (one red and one yellow) in the beginning of the road and one line (yellow) in the end. In case of both Bottom IR sends 1, we have 2 local variables nValue and oValue, we sum both Bottom IR result and place it into nValue variable and place old result of nValue into oValue in order to detect switching

```
if (oValue < 2 && nValue == 2) {
            count++;
            if(count > 2) {
                setCommand(BOTTOM_IR_PRIVILEGE_LEVEL, STOP);
            }
        }
```

then send command to stop or ignore depending on count variable. count is also local variable to count the number of crossed lines.

Top IR and Ultra Sonic

When Top IR find something, it sends 0 and 1 otherwise. Logic is exactly the opposite to Bottom IT, if left IR send 0 turn right and left if right IR send 0. In theory Ultra Sonic must to detect obstacle in front of the car and send the distance between the obstacle and car. But in real world, we forced to disassemble Ultra Sonic and use only Top IR sensors because it does not work properly. Any object near the road, even Ultra Sonic of First obstacle with crazy person can trigger Ultra Sonic of the car or it can miss too low obstacle in the end of the road. And not only our team faced with this problem. Many teams also disassemble Ultra Sonic and used only Top IR sensors for obstacle detect.

For obstacle avoiding we again used counter to differ first obstacle with crazy person when car must stay until it disappears and last one when it must go

round the obstacles. To choose the side, we switch variable cache when car finds something.

```
if(cache == LEFT) {
	cache = RIGHT;
} else if(cache == RIGHT) {
	cache = LEFT;
}
```

In the end, we want to mention that we faced with problem that needed more time to resolve. Raspberry rebooting was our main problem. Car start`s rebooting (restart of OS) process even if does not do anything. Changing of raspberry and batteries did not helped.

We also change OS and rewrote the code but problem was not resolved. Only our team has this problem and even in Internet we did not find proper solutions.

5. Summery

We have searched and learned a lot of information depending to our project. During this course we created our “smart car”. However, due to hardware defects, our raspberry have rebooted several times. We are sure that if we have not such unusual problems, we will have shown better result in final road. In the folder “final_code” you can find: “planB.cpp” and “planBdemo.cpp” which we made if our “planA.cpp” loads the processor or cause problems with connection libraries.