**Final system components summary**

**Components summary**

For electronic components, during the testing session, we used the esp32 microcontroller for data processing. Motor driver boards for signal transmission from esp32 to motor encoders. Two motors and corresponding encoders. Battery for powering the motor. Six Line sensors and sonar sensor for detection of environment. Initially, it is our responsibility to implement the chassis and PCB layout for wiring. According to the latest arrangement of the second semester, the final testing will adopt the uniform Buggy for racing, and we will simulate the application of all the electronic component and data processing with the Puzzlebot platform. Therefore, it is important to figure out the performance of these components.

During the whole racing journey, the sonar sensor performed very well, which contributed to turning on the flat after the ramp. The Buggy was able to turn around and continued to follow the line. On the other hand, the Line sensor was also performed well but incompletely. Because when our Buggy finished turning around, it stopped right at the beginning of the downslope. Since we chose to run the Buggy in one go and we did not have a second chance to fully demonstrate the Buggy, we cannot continue the test. From what the Buggy behaved, we guessed the Buggy didn't detecting line after it begins to go down the slope because only if it didn't detect the line, the Buggy would stop, which was designed for the final stop at the end of the track. Therefore, our line sensor performed not very well. For motors and encoders, it didn't perform very well because at the first attempt in the morning, the Buggy run slowly at a flat track which is line following section and after that, it couldn't climb up the ramp. After discussion, we have a reasonable explanation which is our PWM value allocated to motor is too small. Therefore, we turned up the value. On the second attempt, it performed well in the first section, but it stops at downslope. All of these indicated that we performed the line sensor part not very well. However, we still worked well in sonar parts due to the smooth turning.

**Software/control changes from S1 design:**

Some important changes had been applied in the semester two design. We designed our chassis and PCB in the first semester and designed a tailor-made control scheme to think and simulate the envisaged sensor arrangement fully. Additionally, we came up with unique ideas, such as adding a sonar sensor to the Buggy to aid control.

The basic framework and thoughts were established based on the objectives of the first semester and awaiting testing in the second semester. However, due to the COVID-19, we did not complete some of the proposed designs in the first semester. For example, we did not have the opportunity to implement a control algorithm for a two-row sensor structure and test its stability. Some ideas were retained, such as add a sonar to the Buggy, which became standard in the second semester. Also, we introduced MATLAB simulations, which allowed us to do relatively free simulations on the computer, so we added some new designs that were never mentioned in the first semester:

We added speed control to the Buggy in addition to turning control. i.e. using a second PID algorithm that takes the speed value output from the first PID algorithm for turning as an input value, coupling the three parameters of the PID to let the Buggy accelerate and deaccelerate smoothly, without losing track when a large turn or over-consuming power when a slightly turn had been compensated. The second primary purpose is to give the necessary speed correction when the Buggy goes up or down the slope. We use the "if statement" to let Buggy judge the situation of going up and down. When the difference between two encoder values exceeds the threshold value, the program will deaccelerate or accelerate the Buggy, which depends on the difference is positive or negative. This design is in order to ensure the speed of the Buggy, try to eliminate the effect of inclination on speed.

We had also developed a plan for when the Buggy accidentally run off the track. When the Buggy is on the white line, the output value of the PID is retained by an additional variable, "PID\_backup", which is always up to date. Once the Buggy is off the track and the white line cannot be detected by six sensors, the program will determine if it is a line break, turning point, or run off the track, in order. When the program thinks the car has accidentally run off the track, it reads the value of the encoder on both sides of the Buggy when it was last on the white line and compares it to determine whether the car "wants" to turn right or left, and uses the PID\_backup to allow the car make a significant radius turn until the car can find the track again. However, we did not include this function in the TD4 assessment, so this design was only verified in MATLAB.