Robocup Notebook

Robocup 2023 – Frederick Sun Yuki Wu

# Design Process

## Composition of Robot

The composition of our robot is extremely important, as it determines how all the components fit together, what software we need to be able to program, and how different things are needed to sustain a closed-loop feedback system for a task as intensive as ours.

### Design Material

When building the robot, the basic material needs to be considered. We are aiming for a material that fits:

* Something strong and not brittle, can handle excessive forces.
* Can be easily modified so many different iterations can be tested.
* How readily available it is to our team, in New Zealand, and the relative cost of it.

The options are:

#### Nylon

Nylon is extremely lightweight and low in density, only just 1.14g/cm^3. This would be advantageous to use in higher quantities as it is extremely light and has no significant impact on the design structure of our robot.  
  
Furthermore, Nylon has a great compressive force resistance, with it's Young Modulus, *E*, at 2.7GPa. This would allow us to hold all the heavy components without issue and 3be able to support varying loads.  
  
However, working with Nylon is hard and dangerous. Using Nylon would usually leave a trace of hard, sharp residue after cutting, which poses serious safety concerns.

#### FibreGlass Boards

Fibreglass is extremely strong with a Young Modulus of up to 87 GPa, significantly more resistant to damages that may occur, such as snapping or cracking when being handled. This is one of the main advantages.

It is relatively inexpensive, and easy to work with. However, the execution of creating small, precise hole cut outs is questionable. Furthermore, it requires the use of a binding material, through resin, which would only further complicate the matter.

Another concern is its density. At roughly 48g/cm^3, this is significantly heavier than that of the other materials, and would create extra strain on the motors, which is unneeded for.

#### PLA – 3D printing

#### abs – 3D pRINTING

### Wheel

### Microcontroller

### Claw Design

### Dump Design

## Line Following

### Closed Loop system

The line following process is a complex one: involving many different techniques. We aim to use a closed-loop information feedback system to accurately control our robot and its position within the course. This will improve its accuracy and reliability.

Given the inputs v, for colour sensor inputs, and t, for time, we can use the PID controller to determine the accurate output as a function of both inputs, o(v, t).



This formula describes how the output given time and the sensor inputs can be determined.

e(v,t) is the error difference between two sides of the colour sensors rack, as an error of 0 would indicate that the colour sensors are reading the same value, therefore the same colour, which is most likely both white.

describes the sum of all errors since the controller has started to run. Gives indication to how long an error has been occurring with respect time, showing how this must be a longer turn. This would only work if the error would go negative, otherwise there would be a systematic error to turn whichever way was described as positive.



describes the rate of change of error, given the smallest change in t, which is the last reading. This would give indication as to how rapidly the error is changing with respect to time, meaning any given turn is sharper, hence needing more power.



To properly implement this into our system, we would need to use modified values, as the raw values are much too large to work with for motors. Through 3 different constants,



### Limitations

#### Integral

Using this method does come with limitations, as this is not applicable for our scenario. Given that the error may range between -400 to 400, suggesting that the integral would change dramatically. This means that there would be a high impact relatively, compared to the other parts of the system.

The common solution would have been to lower to KI value. We usually lower it to 10^-4, where there isn't any more impact on the turning to be noticeable. This comes with its drawbacks – the integral is not being properly utilized, and hence we lose out on a major component of the program.

Another issue is when there are multiple turns facing the same way. If the integral takes from 0 to t, current time, then all the past errors will be remembered, and hence if there are consecutive left or right turns, then there would be a bias to turn left or right as the integral value would be high.

This can be seen through an example,

Suppose we have these boards: all consecutively lined up (Although this wouldn't happen in a real scenario, filler boards can be added, achieving the same effect).



Imagine the error as we proceed through these boards, the therefore the sum of errors as integral (Taking a left error as positive, right error as negative)



You can see that after all these turns, there's a clear positive integral. Assuming that the size of errors is consistent for both left and right turns, then we would have been left with a large, positive integral after these turns. Now imagine if the next turn was as follows:



We have a bias to turn left because of the previous turns, and therefore when trying to turn to the right, the proportional and derivative components will have their impacts reduced as the integral still believes that it is trying to turn to the left.

This may lead to the robot missing/underperforming when trying to turn in the opposite direction without buildup. These situations are highly likely, seeing how large the errors can get to, and depending on the curve/sharpness of the turn, how much is stored inside the integral function.

However, we can take advantage of this, instead of simply removing it's large impact on the controller. If we can identify the start and end of a turn, then we can apply the integral to said timestamps:

This means that we can properly take the integral of any turn, and given how long the error lasts, adjust for a higher and sharper turn.



#### Derivative

The derivative measures the rate of change of error. This can be achieved through measuring the last error, and current error, giving the cheapest equation possible to solve for gradient. This is because e(v, t) does not follow a standard formula, to give predictable results, as it relies on the error function which varies a lot with time.

Furthermore, the baud rate of the Arduino in our program is at 115200 bits per second, and the average program cycle is around \_\_\_, which is quick. This results in small values of change between cycles and readings of error, hence the derivative function would have little impact, as it is many small changes over a longer period, compared to a big change over small period, leaving a sharp gradient to be calculated.

However, we can use an artificially delayed function, so instead of taking the derivation between the smallest possible unit of time (between two program cycles), we take an arbitrary one between two different program cycles some time apart.

This lowers the accuracy of the gradient for any given point, and therefore has bigger variations, which would help the robot follow the line as big changes are impactful and may tell us more about the line it's following.

For example:



This seems great on paper but does not account for the end of a turn, where the robot may overcompensate for how much to turn by. We can solve this through incorporating both methods of finding the gradient. Applying the second formula when the change in error spreads from 0, and the first formula when the error returns to 0.

## Intersection Handling

## Obstacle Detection

## Rescue Kit Detection

## Rescue Zone

# Robot Design Iterations

# Daily Entries