Drive Part Report

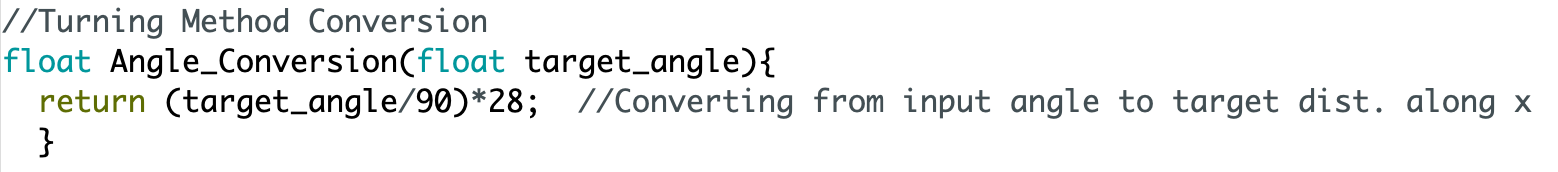
The drive subsystem will allow the rover's movements and communication with control to receive instruction, execute it correctly and send signals back. It will also send the correct distance measurements to the command to allow the calculation of the coordinates. Thus, the drive subsystem could achieve four main tasks: creating movements for the rover, controlling the rover's position, controlling the rover's speed, and measuring the distance travelled by the rover in x and y. In addition, some other extra features could be designed to extend the functionalities of the rover and better enable the user to monitor the rover's status.

Five movements are designed for the rover by setting the wheels to a corresponding low or high state: moving forward, moving backwards, turning left, turning right and stop. It uses the H-bridge circuit to allow DC motors to complete those actions. By achieving moving forward action, the right wheel state will be set to low, and the left wheel state will be set to high while at the same time the H-bridge needs to be enabled and set the PWM signals to be high. The Arduino code for moving forward action is shown as follows:

Text, letter

Description automatically generated

The same method could be applied to moving backward action, whereas the right wheel state will be set to high and the left wheel state will be set to low. Moving forward or backward action required the rover's distance changing along the y-direction. Setting the target distance in the y-direction can make the rover move towards a target position. For the turning left and turning right actions, the wheels state can be both set to be low or high. The distance changes along the x-direction, which equals the circumference covered by the rover's optical sensor while turning a certain angle where the circle's radius is the distance between the optical sensor to the midpoint along the rover shaft is 15cm. However, because two wheels are not outputting the same amount of power, the circle radius would give the correct angle is 17.8cm. The Arduino code for angle conversion function is shown as follows:



The stop instruction could be implemented by disabling the H-bridge and set the PWM signal feeding into the left and right wheel pins to low.

The position control could be achieved by designing a closed-loop PI controller. The control algorithm takes the error between the current position and the previous position. The previous position is considered the place that the rover stopped after executing the previous instruction. Suppose this error is lying between the range for moving forward or backward instructions and for turning left or turning right instructions, the rover would stop. Otherwise, the rover would either be keeping executing the current instruction or doing the reverse movement, and after some oscillations, the rover would finally stop at the target position. The Arduino code for moving forward action is shown as follows:

Graphical user interface, text, application, email

Description automatically generated

Provided 10 seconds to let the rover oscillate and implement the position control until the rover successfully stopped at the position within the error range provided. When the rover successfully executed the instruction, the rover will send back a success signal (InstructionCompleted = 1) to the command to indicate the rover is ready to receive the next instruction. The Arduino code is shown as follows:

Graphical user interface, text, application, chat or text message

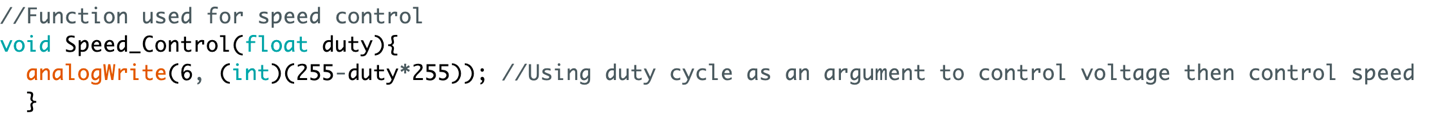
Description automatically generated

Given 20 seconds to allow sufficient time for the rover to finish each instruction, if the rover failed to execute the instruction within this period, the rover would stop and return a failed signal (InstructionCompleted = 0) along with the stop position to command. The Arduino code is shown as follows:

Graphical user interface, text, application, chat or text message

Description automatically generated

It allows the user to monitor the rover's status and let the rover execute the instructions one by one without any interruption. The reason for choosing to design a PI controller is proportional controller could achieve proportional controlling. However, the disadvantage is that the settling time is longer due to the steady-state error. By adding the integral controller, it will get rid of this steady-state error. By tuning the parameters for and , the best results obtained is = 1 and = 1. By creating a closed-loop control strategy, position control and speed control can be linked together. The speed control is achieved by setting the PWM signal into the SMPS board duty cycle to control the output voltage applied to motors. The Arduino code for the speed control function is shown as follows:



By taking the position error in either x or y direction as input and send into the PI control function, the output can be used as an argument to input to the speed control function. After some trials, the duty cycle input function is: , where 0.4 is the minimum to activate the rover. Because of the presence of an integral controller, the error will get accumulated as time increases. If the rover stays at a place for a long time, the error will accumulate, and the rover would speed up and leave that place. In practice, the rover will initially accelerate due to the significant initial error. As the rover approaches the target, the rover will decelerate, which means the measurements taken by the optical sensor would be more accurate. This method effectively reduces the number of oscillations and will result in a more accurate rover performance. The relationship between the duty cycle and speed can be created as a look-up table. By using the formula: , the look-up-table is as follows:



It will create an advanced feature that when the rover is climbing up a slope, the speed will be different when the rover travels at a flat surface. Using this look-up table and characterising the terrain's height will indicate on the map plotting the slope that the rover travelled. According to the look-up table above, the speed is around 11cm/s when the duty cycle is larger than 0.6. By creating a power-saving mode, the duty could be fixed between 0.4 to 0.8.

The optical sensor will measure the distance of the rover travelled in the x- and y-direction. In order to obtain more accurate measurements, the measuring data type could be changed from integer to float. The turning angle and travelling distance along y-direction can be sent to command through control to calculate the coordinates. Using the turning angle to define the orientation that the rover is facing, and it is assumed that the rover head is initialling facing north. If any turning instructions are executed, the orientation will change. Then, moving forward or backward, instructions will either change the x-coordinate or y-coordinate. For instance, if the rover faces north or east, moving forward, instruction will change y-coordinate or x-coordinate positively. If the rover is facing south or east, moving forward, instruction will change y-coordinate or x-coordinate negatively.

The drive subsystem also plays a crucial role in receiving instructions and send a finish signal to command through control. When the control sends instructions to drive, a decoding function will take the action word and target angle or target distance in the y-direction. The action word is defined as 'F', 'B', 'L', 'R', 'S', 'X', which indicates moving forward, backward, left turning, right turning, stop, and emergency stop movements. The target arguments will be taken to implement the position and speed control. The Arduino code for receiving instruction from the control and decoding moving forward instruction is shown as follows:

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

The emergency stop action is an advanced feature when an emergency happens in front of the rover, but vision failed to detect that. In that case, the emergency stop command would immediately stop the rover, send an 'emergency stop' message back to the command, and treat the current instruction to be failed to execute. The emergency stop is differentiated to stop, because for the stop instruction, if the rover correctly executed, this would set the InstructionCompleted signal to high. However, the emergency stop will set the InstructionCompleted signal to low. The Arduino code for decoding the emergency stop function is shown as follows:

Text

Description automatically generated with low confidence

Another advanced feature is that by giving a pause and un-pause command ('P' and 'U' as action word), the rover would stop and let the vision take pictures to detect the distance between the rover and the obstacle. After the un-pause command is given, the rover will continue executing the current instruction. The Arduino code for this feature is shown as follows:

Text

Description automatically generated with low confidence