Checkpoint #2 Report

[EECN30169] Mobile Robot 2022

Student ID:310611008 **Name:** 張祐誠 **Date: Oct 24, 2022**

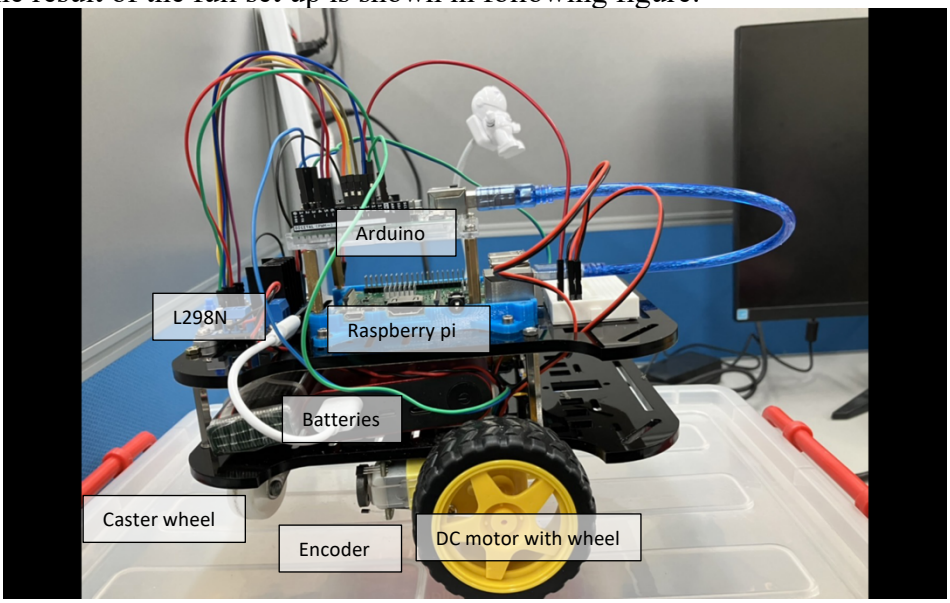
1. Purpose:

The purpose of this checkpoint required us to:
1. Assembly the mobile robot and control by ROS through raspberry pi and Arduino.
2. Control the DC motors by encoder signal to make the robot moving in different direction. In the moving forward task, it is required to make the robot walk as straight as possible.

2. Description of Design:

A. Construction of mobile robot:

1.The result of the full set up is shown in following figure:



B. Design of control procedure:

1. We first made a custom message type called “target_speed”, which the msg file is composed by two integer elements. As shown in figure, the two elements store the speed command given by user.

```
jeffchang@jeffchang-respi3:~/catkin_ws$ rosmsg show target_speed
[checkpoint_2/target_speed]:
int32 left_speed
int32 right_speed
```

2. Then a publisher was written to input the user command. This step will accept the velocity of two wheels given by the user respectively and compose into a single msg file. Notice that the msg file is build and compiled in the previous step. The msg contained the user’s input will then published to the Arduino board, the serial node.

```python
def publisher():
    pub = rospy.Publisher('target', target_speed, queue_size=10)
    rospy.init_node("publisher")
    while not rospy.is_shutdown():
        left_v = int(input("left wheel speed:"))
        right_v = int(input("right wheel speed:"))
        msg = target_speed()
        msg.left_speed = left_v
        msg.right_speed = right_v

        pub.publish(msg)
        rospy.sleep(1)
```

3. Then the Arduino take the rest of work. We first initialized a subscriber to subscribe the command sent from the previous step. Some variables will be set to zero for the PID controller at the same time.

```
void messageCb(const checkpoint_2::target_speed& input_msg){
  pre_error_R = 0;
  acc_error_R = 0;
  pre_error = 0;
  acc_error = 0;
  left_command = input_msg.left_speed;
  right_command = input_msg.right_speed;
}
ros::Subscriber<checkpoint_2::target_speed> sub("/target", messageCb);
```

4. Another important part is the initiation of the encoder. An interrupt was set up in this step. The number 0 was referred to the pin 2, which is the special pin on Arduino uno that come with the interrupt function. Similarly, pin 3 was referred by number 1.

```
void encoderInit(){
  direction_L =  true;
  pinMode(Ec0A, INPUT);
  pinMode(Ec0B, INPUT);
  attachInterrupt(0, wheelSpeed_L, CHANGE);
```

5. The encoder was worked by two hall sensors. The sensors send different phase of signal. The one attached to the interrupt pin will trigger the interruption and the other pin will read the rise signal. The sequence of the signals will determine the direction of rotation, and the numbers of pulse will accumulate to represent the rotation speed. That is, the output of the encoder represent how fast the wheel is rotating.

```
void wheelSpeed_L(){
  int Lstate = digitalRead(Ec0A);
  if((Ec0A_last == LOW) && (Lstate == HIGH)){
    int val = digitalRead(Ec0B);
    if(val == LOW && direction_L){
      direction_L = false;
    }
    else if(val == HIGH && !direction_L){
      direction_L = true;
    }
  }
  Ec0A_last = Lstate;

  if(!direction_L) duration_L--;
  else duration_L++;
}
```

6. The PID controller will then take into part in this step. The Kp, Ki and Kd variable was set up for each wheel separately by trial and error. The working principle of PID controller is well known and used widely nowadays. P controller can reduce the steady state error, I controller canceled the steady state error, and D controller suppress the overshoot caused by I controller.

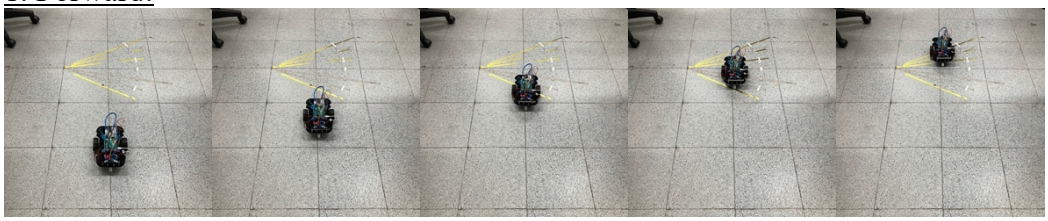
```
float pid_control(int ref_speed, int current_speed){
  float Kp = 7;
  float Ki = 0.03;
  float Kd = 0.01;
  float error = ref_speed - current_speed;
  acc_error += error*0.05;
  float output = Kp*error + Kd*(error - pre_error) + Ki*acc_error;
  pre_error = error;

  return output;
}
```

7.Lastly, the DC motor was controlled to trace the reference input by user.

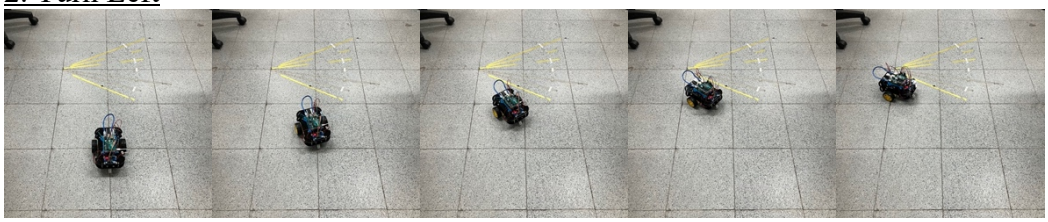
```
void activate(int current_speed_L, int current_speed_R){
 if(left_command>0 && right_command>0){
  float pwm_L = sat(left_command + pid_control(left_command, current_speed_L));
  float pwm_R = sat(right_command+ pid_control_R(right_command, current_speed_R));
  forward((int)(pwm_L+0.5), (int)(pwm_R+0.5));
 }
 else if(left_command==0 && (right_command-supply) ==0){
  freeze();
 }
 else if(left_command<0 && right_command<0){
  float pwm_L = sat(-left_command + pid_control(-left_command, -current_speed_L));
  float pwm_R = sat(-right_command+supply_b+ pid_control_R(-right_command+supply_b, -current_speed_R));
   backward((int)(pwm_L+0.5), (int)(pwm_R+0.5));
 }
```

3. Result

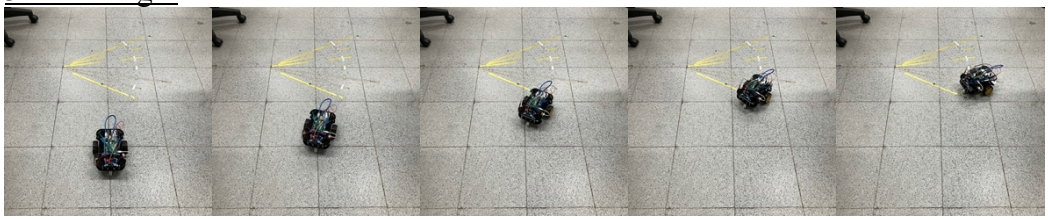
1. Forward:



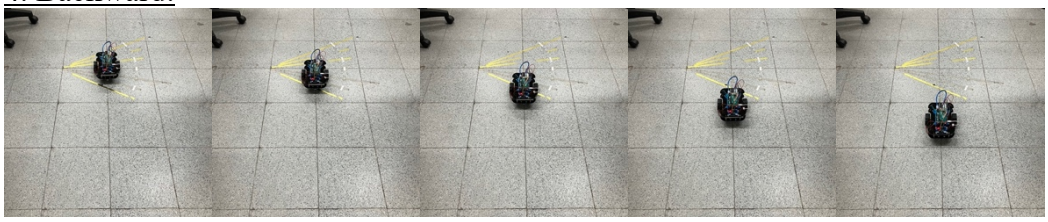
2. Turn Left



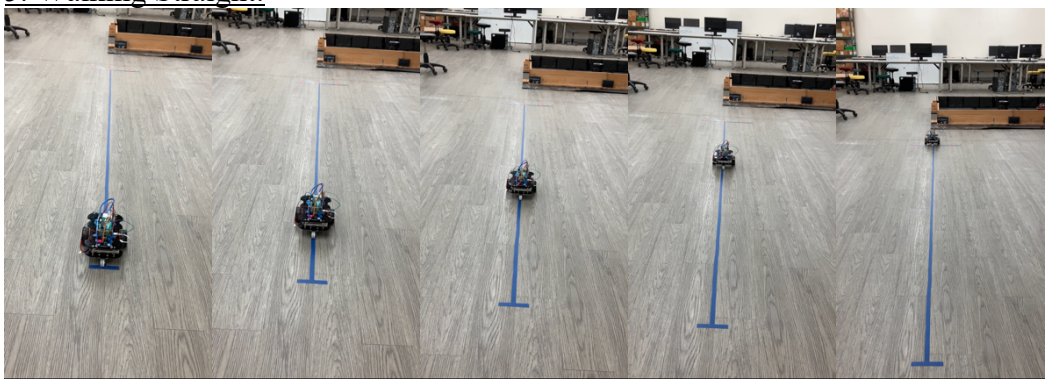
3. Turn Right



4. Backward:



5. Walking Straight:



4. Discussion

1. We wrote a publisher in Arduino to publish the encoder data. By the $*rostopic echo /encoder_data* command, we are able to monitor the angular velocity of each motor.

```
void publish_data(int data_L, int data_R){
  output_msg.left_speed = data_L;
  output_msg.right_speed = data_R;
  output.publish(&output_msg);
}
```

2. By the output result from the previous point, we can observe how does the encoder effect the motor speed. In the following figure is the encoder data when the motor is working without controller. It is obvious that the two motor were not spinning at the same speed. And, as expected, the robot will tend to turn on one side.

```
---
left_speed: 108
right_speed: 101
---
left_speed: 108
right_speed: 100
---
left_speed: 107
right_speed: 100
---
left_speed: 108
right_speed: 101
---
left_speed: 108
right_speed: 100
---
left_speed: 108
right_speed: 100
---
left_speed: 107
right_speed: 100
---
left_speed: 108
right_speed: 100
```

The following figure is the encoder data when the motor is working with PID controller. We can easily tell that the result is more stable and the mobile is walking forward almost perfect. In the demo, our robot only shift 0.9 mm to the right after walking 3 meter forward.