Checkpoint 2　Report

[EECN30169] Mobile Robot 2022

Student ID:　312512063　Name: 張祐維　Date: 2023/10/30

• Purpose:

The purpose of this project is to design and implement a control system that utilizes a Raspberry Pi and an Arduino to control two DC servo motors. The primary goal is to achieve precise speed control of these motors using a PID (Proportional-Integral-Derivative) control algorithm. This report details the design, results, and discussions related to this project.

• Description of Design:

Raspberry Pi: The Raspberry Pi is used to send speed control commands to the Arduino. It acts as the main controller for the system.

The Raspberry Pi runs a Python script that initializes a ROS (Robot Operating System) node and publishes speed commands for the left and right motors. Users can input speed values that are sent to the Arduino for motor control.

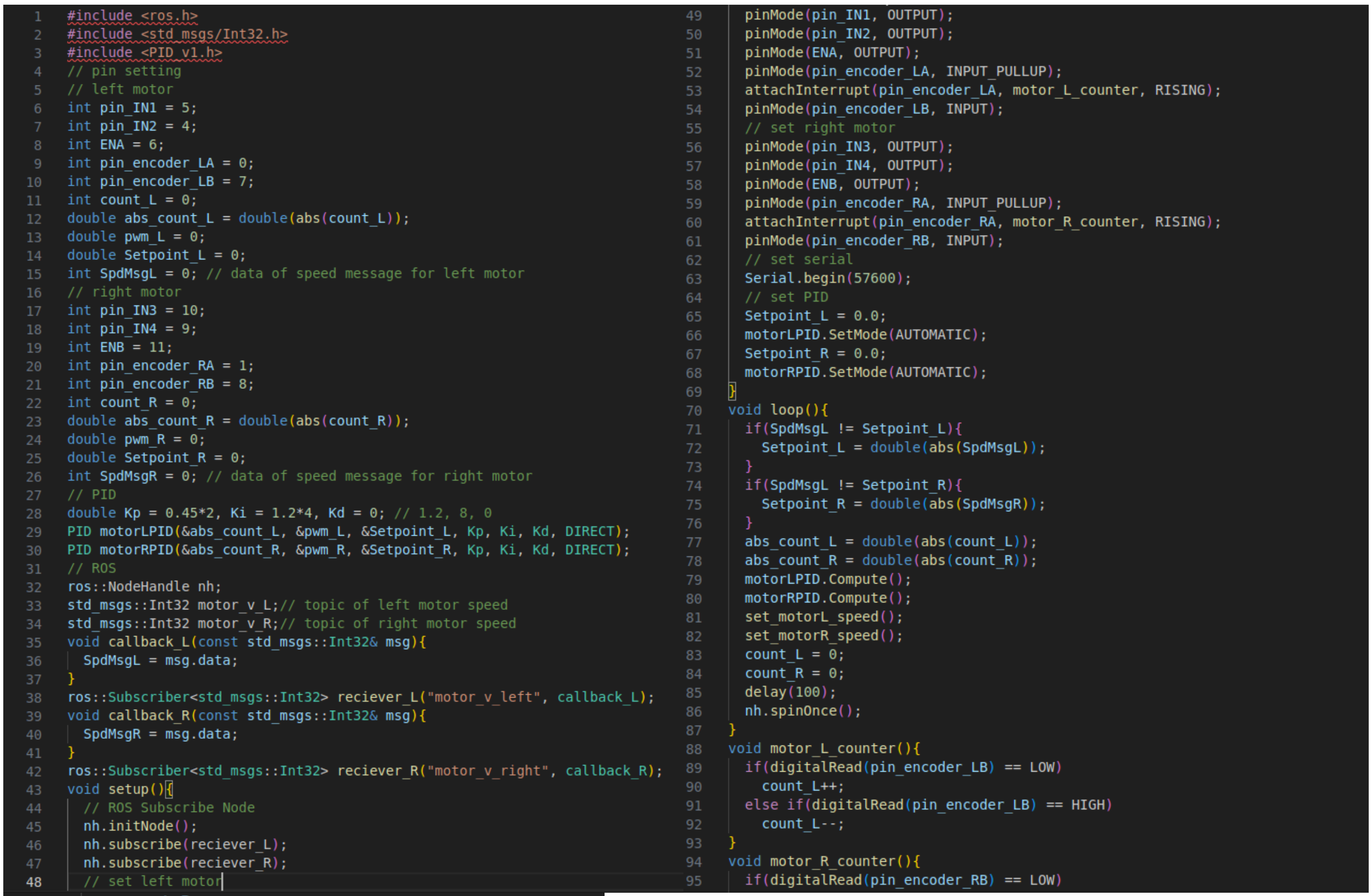
Arduino Uno: The Arduino serves as the microcontroller that directly controls the DC servo motors based on the commands received from the Raspberry Pi.

The Arduino code initializes the necessary pins, subscribes to speed commands from the Raspberry Pi via ROS, calculates the required PWM signals for the motors, and implements the PID control algorithm to maintain the desired motor speed.

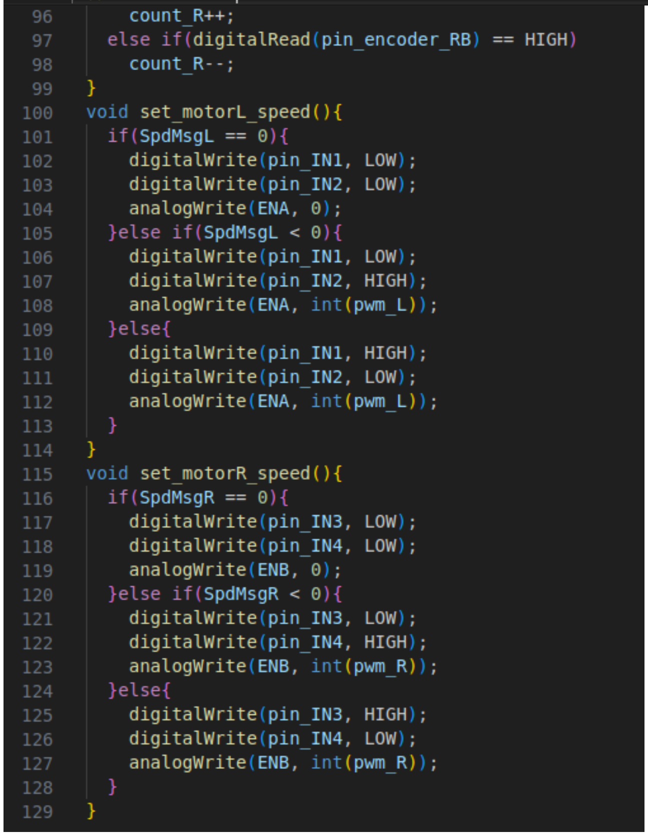
DC Servo Motors: Two DC servo motors are employed in the system. These motors are used for various applications where precise control over the motor's rotational speed is necessary.

Rotary Encoders: Rotary encoders are used as feedback devices to measure the motors' actual speed and provide feedback to the control system.

Arduino’s code

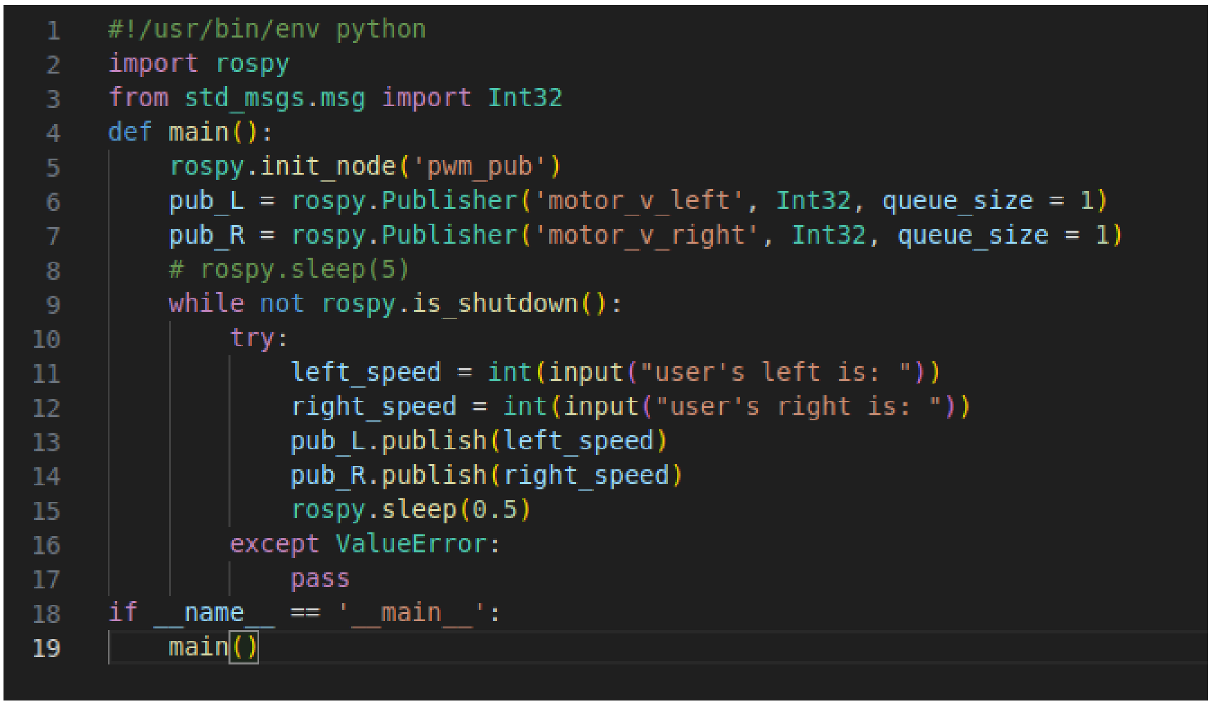


```
#include <ros.h>
#include <std_msgs/Int32.h>
#include <PID_v1.h>
// pin setting
// left motor
int pin_IN1 = 5;
int pin_IN2 = 4;
int ENA = 6;
int pin_encoder_LA = 0;
int pin_encoder_LB = 7;
int count_L = 0;
double abs_count_L = double(abs(count_L));
double pwm_L = 0;
double Setpoint_L = 0;
int SpdMsgL = 0; // data of speed message for left motor
// right motor
int pin_IN3 = 10;
int pin_IN4 = 9;
int ENB = 11;
int pin_encoder_RA = 1;
int pin_encoder_RB = 8;
int count_R = 0;
double abs_count_R = double(abs(count_R));
double pwm_R = 0;
double Setpoint_R = 0;
int SpdMsgR = 0; // data of speed message for right motor
// PID
double Kp = 0.45*2, Ki = 1.2*4, Kd = 0; // 1.2, 8, 0
PID motorLPID(&abs_count_L, &pwm_L, &Setpoint_L, Kp, Ki, Kd, DIRECT);
PID motorRPID(&abs_count_R, &pwm_R, &Setpoint_R, Kp, Ki, Kd, DIRECT);
// ROS
ros::NodeHandle nh;
std_msgs::Int32 motor_v_L;// topic of left motor speed
std_msgs::Int32 motor_v_R;// topic of right motor speed
void callback_L(const std_msgs::Int32& msg){
  SpdMsgL = msg.data;
}
ros::Subscriber<std_msgs::Int32> reciever_L("motor_v_left", callback_L);
void callback_R(const std_msgs::Int32& msg){
  SpdMsgR = msg.data;
}
ros::Subscriber<std_msgs::Int32> reciever_R("motor_v_right", callback_R);
void setup(){
  // ROS Subscribe Node
  nh.initNode();
  nh.subscribe(reciever_L);
  nh.subscribe(reciever_R);
  // set left motor
  pinMode(pin_IN1, OUTPUT);
  pinMode(pin_IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(pin_encoder_LA, INPUT_PULLUP);
  attachInterrupt(pin_encoder_LA, motor_L_counter, RISING);
  pinMode(pin_encoder_LB, INPUT);
  // set right motor
  pinMode(pin_IN3, OUTPUT);
  pinMode(pin_IN4, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(pin_encoder_RA, INPUT_PULLUP);
  attachInterrupt(pin_encoder_RA, motor_R_counter, RISING);
  pinMode(pin_encoder_RB, INPUT);
  // set serial
  Serial.begin(57600);
  // set PID
  Setpoint_L = 0.0;
  motorLPID.SetMode(AUTOMATIC);
  Setpoint_R = 0.0;
  motorRPID.SetMode(AUTOMATIC);
}
void loop(){
  if(SpdMsgL != Setpoint_L){
    Setpoint_L = double(abs(SpdMsgL));
  }
  if(SpdMsgL != Setpoint_R){
    Setpoint_R = double(abs(SpdMsgR));
  }
  abs_count_L = double(abs(count_L));
  abs_count_R = double(abs(count_R));
  motorLPID.Compute();
  motorRPID.Compute();
  set_motorL_speed();
  set_motorR_speed();
  count_L = 0;
  count_R = 0;
  delay(100);
  nh.spinOnce();
}
void motor_L_counter(){
  if(digitalRead(pin_encoder_LB) == LOW)
    count_L++;
  else if(digitalRead(pin_encoder_LB) == HIGH)
    count_L--;
}
void motor_R_counter(){
  if(digitalRead(pin_encoder_RB) == LOW)
```



```
    count_R++;
  else if(digitalRead(pin_encoder_RB) == HIGH)
    count_R--;
}
void set_motorL_speed(){
  if(SpdMsgL == 0){
    digitalWrite(pin_IN1, LOW);
    digitalWrite(pin_IN2, LOW);
    analogWrite(ENA, 0);
  }else if(SpdMsgL < 0){
    digitalWrite(pin_IN1, LOW);
    digitalWrite(pin_IN2, HIGH);
    analogWrite(ENA, int(pwm_L));
  }else{
    digitalWrite(pin_IN1, HIGH);
    digitalWrite(pin_IN2, LOW);
    analogWrite(ENA, int(pwm_L));
  }
}
void set_motorR_speed(){
  if(SpdMsgR == 0){
    digitalWrite(pin_IN3, LOW);
    digitalWrite(pin_IN4, LOW);
    analogWrite(ENB, 0);
  }else if(SpdMsgR < 0){
    digitalWrite(pin_IN3, LOW);
    digitalWrite(pin_IN4, HIGH);
    analogWrite(ENB, int(pwm_R));
  }else{
    digitalWrite(pin_IN3, HIGH);
    digitalWrite(pin_IN4, LOW);
    analogWrite(ENB, int(pwm_R));
  }
}
```

Raspberry PI’s code



```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
def main():
    rospy.init_node('pwm_pub')
    pub_L = rospy.Publisher('motor_v_left', Int32, queue_size = 1)
    pub_R = rospy.Publisher('motor_v_right', Int32, queue_size = 1)
    # rospy.sleep(5)
    while not rospy.is_shutdown():
        try:
            left_speed = int(input("user's left is: "))
            right_speed = int(input("user's right is: "))
            pub_L.publish(left_speed)
            pub_R.publish(right_speed)
            rospy.sleep(0.5)
        except ValueError:
            pass
if __name__ == '__main__':
    main()
```

- ## Result

The implementation of the control system yielded several notable results:

Precise Motor Control: The system successfully achieved precise control over the rotational speed of the DC servo motors. The PID control algorithm adjusted the motor speed in response to speed commands, and the feedback from rotary encoders ensured accurate speed control.
Responsive Communication: The communication between the Raspberry Pi and Arduino, facilitated by ROS, allowed for real-time speed adjustments. Users could input speed commands, and the motors responded quickly and accurately.
Tuning Possibilities: The PID controller's parameters, such as Kp, Ki, and Kd, can be adjusted to fine-tune the system's response to different operating conditions, making it adaptable for various applications.

- **Discussion**

The successful implementation of this control system demonstrates the feasibility of using Raspberry Pi and Arduino for precise motor control. The combination of a user-friendly Raspberry Pi interface and the real-time control capabilities of the Arduino allows for a wide range of applications, including robotics, automation, and mechatronics.

Additionally, the use of PID control proved effective in maintaining motor speed, but the specific PID parameters (Kp, Ki, Kd) may need to be adjusted to optimize performance for different scenarios.

Furthermore, the system's scalability can be explored by adding more motors and incorporating additional sensors or features to enhance its functionality.

In conclusion, this project highlights the potential of Raspberry Pi and Arduino-based control systems for motor control and serves as a foundation for further developments in the field of automation and robotics. The successful integration of hardware and software components demonstrates the versatility and adaptability of such systems for a wide range of applications.