教育部優化技職校院實作環境計畫
建置跨院系實作場域

智慧新零售-全通路跨域整合體驗中心

物聯網創意應用

課程教材

建置以 ROS 控制的小車

說明

本系列介紹小車軟硬體要求及搭建方法，實現透過 ROS (Robot Operation System)來控制小車運動。

內容

第一篇硬體

說明：介紹相關硬體

硬體清單：

- 樹莓派 2 或 3 代
- 帶編碼器測速 JGA25-371 減速馬達一對
- 小車輪子一對
- 萬向輪一個
- 小車壓克力板底板一套
- Arduino MEGA 2560 R3 開發板一塊：用於控制馬達驅動，接收上位機指令並把感測器
- L298N 馬達驅動板模組
- 12V 充電電池
- 分壓模組

搭建步驟：

第一步：把 JGA25-371 減速馬達和輪子連接，然後和萬向輪一起固定到小車壓克力板底板上。

第二步：考慮 JGA25-371 減速馬達、L298N 馬達驅動板模組、Arduino MEGA 2560 R3、樹莓派和電池之間如何連接。

零件介紹：

- JGA25-371 減速馬達

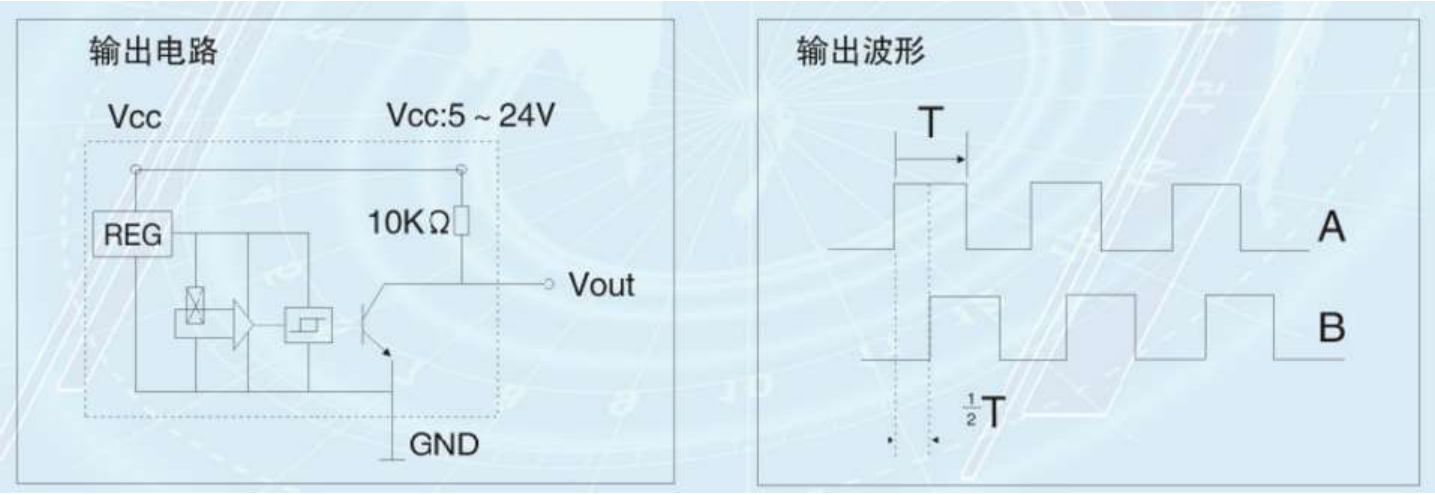
JGA25-371 帶編碼器測速碼盤的馬達由兩部分組成，**直流減速馬達**和**雙通道霍爾效應編碼器**。

直流減速馬達的工作電壓範圍在 6~24V DC，依據在額定電壓 12V 時，每分鐘的空載轉速，JGA25-371 系列馬達分為如下幾個型號：977 RPM、463 RPM、201 RPM、126 RPM、95 RPM、55 RPM、41 RPM、25 RPM、19 RPM、11 RPM、8.6 RPM

型號：JGA25-371 參數表												
電壓 (Voltage)		空載 (No Load)		負載轉矩 (Load Torque)				堵轉 (Stall)		減速器 (Reducer)		重量 (Weight)
範圍 Workable Range	額定 Rated Voltage	轉速 Speed	電流 Current	轉速 Speed	電流 Current	扭矩 Torque	功率 Output	扭矩 Torque	電流 Current	減速比 Ratio	尺寸 Size	g
V	V	RPM	mA	RPM	mA	Kg.cm	W	Kg.cm	A	1:XX	mm	
6-24V	12	977	46	781	300	0.11	1.25	0.55	1	4.4	15	80
6-24V	12	463	46	370	300	0.23	1.25	1.1	1	9.28	17	82
6-24V	12	201	46	168	250	0.53	1.25	2.65	1	21.3	19	84
6-24V	12	126	46	100	250	0.85	1.25	4.2	1	34	21	85
6-24V	12	95	46	76	200	1.1	1.25	5.5	1	45	21	86
6-24V	12	55	46	44	200	1.95	1.25	9.7	1	78	23	88
6-24V	12	41	46	32	150	2.5	1.25	12.5	1	103	23	89
6-24V	12	25	46	20	150	4.2	1.25	21	1	171	25	91
6-24V	12	19	46	15	120	5.6	1.25	28	1	226	25	92
6-24V	12	11	46	8.8	120	9.45	1.25	47	1	378	27	94
6-24V	12	8.6	46	6.8	120	12	1.25	60	1	500	27	96

轉速越快，做出來小車的速度就會越快，但轉速快的小車扭矩就小，載重也隨之變小，並且，在上坡或越過障礙物時就顯得動力不足，建議購買型號為 126 RPM 轉速的馬達。

測速的編碼器是**雙通道霍爾效應編碼器**，它包含一個磁柵和磁敏檢測電路，輸出兩個通道正交相位角為 90 度的方波



編碼器單路每圈脈衝 13 CPR(Counts Per Revolution，每轉脈衝的個數)，由於每圈可以區分為一個上升沿和一個下降沿，另一方面此編碼器具有 A、B 雙路輸出，所以每轉一圈，編碼器的雙路、上下沿總共可以輸出 52 CPR。

那麼小車輪子轉一圈，最多只能產生 52 個脈衝信號嗎？不對，還要考慮減速比，以轉速為 126 RPM 的型號馬達為例，它的減速比為 1:34，也就是說，編碼器測轉 34 圈，小車的輪子才轉一圈。所以，小車輪子轉一圈，可以產生 52*34=1,768 個脈衝信號(就是所謂的解析度)。另外，還可以根據 A、B 雙路輸出的信號，確定輪子是正轉還是反轉。

JGA25-371 馬達和編碼器的接線如下圖，

依照上圖，接線從右到左分別是 M+、M-、Hgnd、Hvcc、HoutA、HoutB。

M+(紅線)：馬達電源+

M-(黑線)：馬達電源-

Hgnd(綠線)：編碼器地線

Hvcc(藍線)：編碼器電源(接 5V 電壓)

HoutA(黃線)：信號 A 輸出

HoutB(白線)：信號 B 輸出

測速編碼的連線

```
#define BAUDRATE    115200
#define LEFT        0
#define RIGHT       1
#define FORWARDS    true
#define BACKWARDS   false

volatile long encoderLeft = 0L;
volatile long encoderRight = 0L;

void initEncoders() {
    pinMode(2, INPUT);
    pinMode(3, INPUT);
```

```
    attachInterrupt(0, encoderLeftISR, CHANGE);
    attachInterrupt(1, encoderRightISR, CHANGE);
}

void encoderLeftISR() {
    encoderLeft++;
}

void encoderRightISR() {
    encoderRight++;
}

long readEncoder(int i) {
    long encVal = 0L;
    if (i == LEFT) {
        noInterrupts();
        // detachInterrupt(0);
        encVal = encoderLeft;
        interrupts();
        // attachInterrupt(0, Code_left, FALLING);
    } else {
        noInterrupts();
        // detachInterrupt(1);
        encVal = encoderRight;
        interrupts();
        // attachInterrupt(1, Code_right, FALLING);
    }
    return encVal;
}

/* Wrap the encoder reset function */
void resetEncoder(int i) {
    if (i == LEFT) {
        noInterrupts();
        encoderLeft = 0L;
        interrupts();
    } else {
        noInterrupts();
        encoderRight = 0L;
        interrupts();
    }
```

```
}

/* Wrap the encoder reset function */
void resetEncoders() {
    resetEncoder(LEFT);
    resetEncoder(RIGHT);
}

void setup() {
    Serial.begin(BAUDRATE);
    initEncoders();
    resetEncoders();
}

void loop() {
    long lval= readEncoder(0);
    long rval= readEncoder(1);
    Serial.print("left: ");
    Serial.print(lval);
    Serial.print("; right: ");
    Serial.println(rval);
    delay(30);
}
```

效果圖：

L298N 馬達驅動模組

驅動電壓 5V~35V，邏輯電壓 5V，內置的 78M05 通過驅動電源部分取電工作，當使用大於 12V 驅動電壓的時候，為了避免穩壓晶片損壞，請使用外置的 5V 邏輯供電。

ENA，ENB：OUT1 和 OUT2 的致能接頭。當 Enable(跳線帽短路，接上)時，高電平表示 OUT1/OUT2 是有效可控制的(如下表，直流電機)，低電平則失效(disabled)；如果馬達需要 PWM 調速，則將跳線帽拔掉，成為斷路。

本模組是 2 路的 H 橋式驅動，所以可以同時驅動 2 顆直流馬達(或 1 顆步進馬達)，致能 ENA、ENB 之後，從 IN1，IN2 輸入高、低信號驅動馬達 1 的轉速和方向，從 IN3，IN4 輸入高、低信號驅動馬達 2 的轉速和方向。

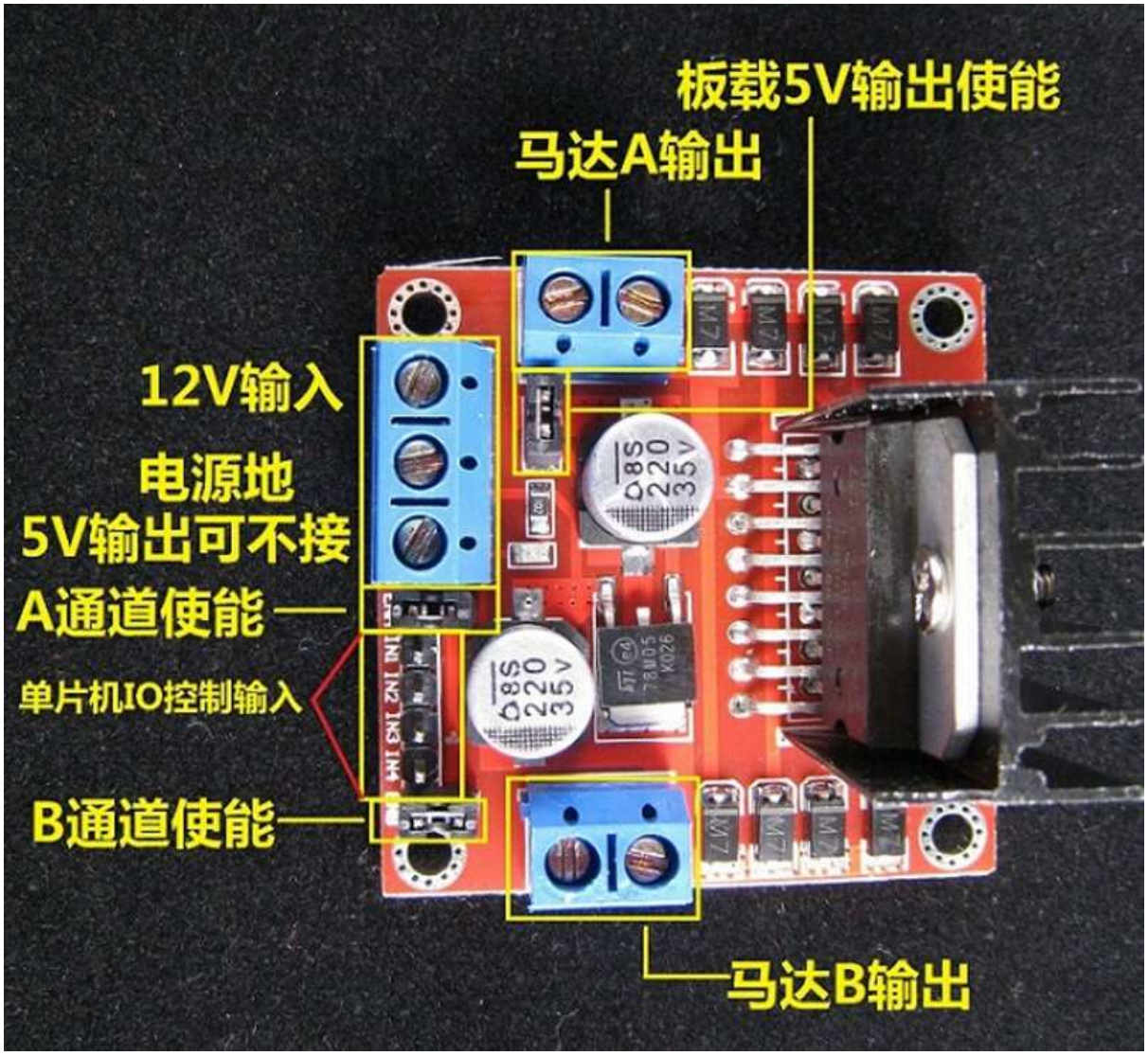
- 驅動直流馬達

直流电机	旋转方式	IN1	IN2	IN3	IN4	调速PWM信号	
						调速端A	调速端B
M1	正转	高	低	/	/	高	/
	反转	低	高	/	/	高	/
	停止	低	低	/	/	高	/
M2	正转	/	/	高	低	/	高
	反转	/	/	低	高	/	高
	停止	/	/	低	低	/	高

- 驅動步進馬達

步进电机	信号输入	第一步	第二步	第三步	第四步	返回第一步
正转	IN1	0	1	1	1	返回
	IN2	1	0	1	1	返回
	IN3	1	1	0	1	返回
	IN4	1	1	1	0	返回
反转	IN1	1	1	1	0	返回
	IN2	1	1	0	1	返回
	IN3	1	0	1	1	返回
	IN4	0	1	1	1	返回

L298N 驅動板介紹：



如果 L298N 和 Arduino 使用不同的電源供電的話，那麼需要將 Arduino 的 GND 和模組上的 GND

連接在一起，只有這樣單片機上過來的邏輯信號才有個參考 0 點。此點非常重要，請大家注意。

L298N 驅動板接線方法

方式 1：ENA/ENB 插帽拔掉，以 IN1、IN2/IN3、IN4 控制馬達轉動方向(正轉或反轉)，另外以 ENA/ENB 則可以控制馬達的轉速。

測試代碼：

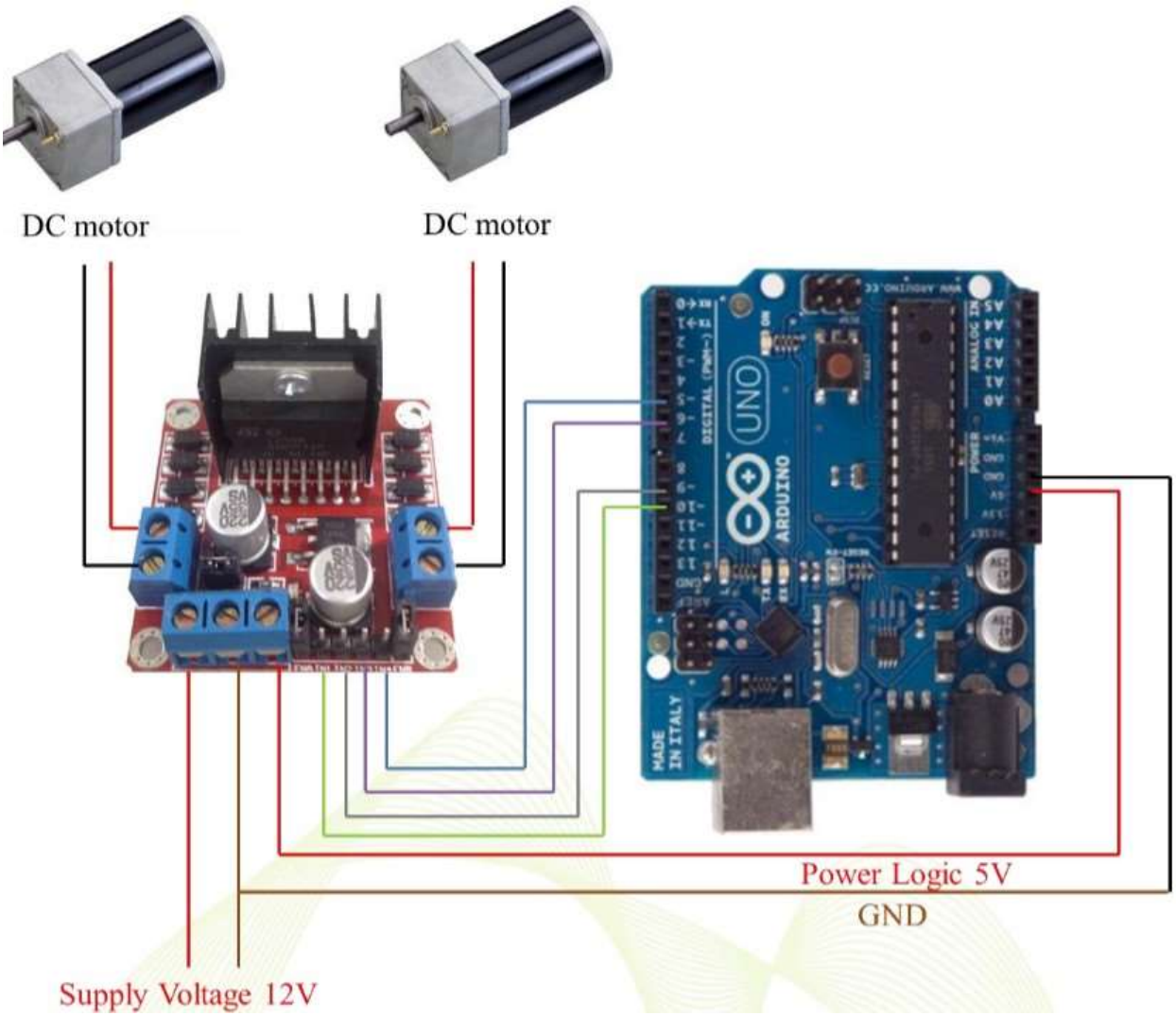
```
// motor A
int enA = 5;
int in1 = 7;
int in2 = 8;
// motor B
int enB = 6;
int in3 = 9;
int in4 = 10;

void setup() {
    // set all the motor control pins to outputs
    pinMode(enA, OUTPUT);
    pinMode(enB, OUTPUT);
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(in3, OUTPUT);
    pinMode(in4, OUTPUT);
}

void demoOne() {
    // this function will run the motors in both directions at a fixed speed

    // turn on motor A
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    // set speed to 200 out of possible range 0~255
    analogWrite(enA, 200);

    // turn on motor B
    digitalWrite(in4, HIGH);
    digitalWrite(in3, LOW);
```

```
    // set speed to 200 out of possible range 0~255
    analogWrite(enB, 200);

    delay(2000);

    // now change motor directions
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    digitalWrite(in4, LOW);
    digitalWrite(in3, HIGH);

    delay(2000);

    // now turn off motors
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
}

void demoTwo() {
    /*
        this function will run the motors across
        the range of possible speeds
        note that maximum speed is determined by
        the motor itself and the operating voltage
        the PWM values sent by analogWrite() are
        fractions of the maximum speed possible
        by your hardware
    */

    // turn on motors
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);

    // accelerate from zero to maximum speed
    for (int i = 0; i < 256; i++) {
        analogWrite(enA, i);
        analogWrite(enB, i);
        delay(20);
```

```
    }

    // decelerate from maximum speed to zero
    for (int i = 255; i >= 0; --i) {
        analogWrite(enA, i);
        analogWrite(enB, i);
        delay(20);
    }

    // now turn off motors
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
}

void loop() {
    demoOne();
    delay(1000);
    demoTwo();
    delay(1000);
}
```

方式 2：ENA 插帽不用拔掉，通過 IN1 和 IN2 輸入高、低信號控制馬達轉動的方向和轉速。

連線圖：



參考：

- [L298N 馬達驅動板介紹]

第二篇軟體

說明

介紹小車需要的軟體

軟體清單：

- Ubuntu 14.04 ➔ Ubuntu 18.04 LTS
- Indigo 版本 ROS 系統 ➔ Melodic 版本 ROS 系統
- Indigo 版本 `ros_arduino_bridge` ➔ `Kinetic` 版本 `ros_arduino_bridge(?)`

前綴程式路徑：

`~/catkin_ws/src/ros_arduino_bridge/`

程式構成：

- 編碼器回饋
- 馬達控制
- PID 演算法
- 主程序

程式更改:

由於 `ros_arduino_bridge` 中定義的編碼器、馬達和本文使用的不相同，所以在編碼器和馬達部分的程式需要比較大的改動。

編碼器

編碼器 1 接線：把 JGA25-371 中編碼器 1 的 HoutA(黃線)和 HoutB(白線)接 Arduino Mega 2560 的 pin2 和 pin3。

工作原理：當輪子轉動，pin2 和 pin3 引腳就會產生 0 號和 1 號中斷，每次產生中斷，根據輪子轉動的方向計數器+1 或-1。

編碼器 2 接線：編碼器 2 的 HoutA(黃線)和 HoutB(白線)接 Arduino Mega 2560 的 pin19 和 pin18，對應的中斷號是 4 和 5 (當然也可接 pin21 和 pin20，對應的中斷 2 和 3)

ros_arduino_firmware/src/libraries/ROSArduinoBridges/encoder_driver.h：

```
long readEncoder(int i);
void resetEncoder(int i);
void resetEncoders();

void initEncoders();
void encoderRightISR();
void encoderLeftISR();
```

ros_arduino_firmware/src/libraries/ROSArduinoBridges/encoder_driver.ino：

```
#include "motor_driver.h"
#include "commands.h";

/* encode */
volatile long left_enc_pos = 0L;
volatile long right_enc_pos = 0L;
int left_rotate = 0;
int right_rotate = 0;

void initEncoders() {
    pinMode(2, INPUT);
    pinMode(3, INPUT);
    pinMode(19, INPUT);
    pinMode(18, INPUT);
    attachInterrupt(0, encoderLeftISR, CHANGE);
    attachInterrupt(1, encoderLeftISR,  CHANGE);
    attachInterrupt(4, encoderRightISR, CHANGE);
    attachInterrupt(5, encoderRightISR, CHANGE);
}

void encoderLeftISR() {
```

```
    if (direction(LEFT) == BACKWARDS) {
        left_enc_pos--;
    } else {
        left_enc_pos++;
    }
}

void encoderRightISR() {
    if (direction(RIGHT) == BACKWARDS) {
        right_enc_pos--;
    } else {
        right_enc_pos++;
    }
}

long readEncoder(int i) {
    long encVal = 0L;
    if (i == LEFT) {
        noInterrupts();
        encVal = left_enc_pos;
        interrupts();
    } else {
        noInterrupts();
        encVal = right_enc_pos;
        interrupts();
    }
    return encVal;
}

/* Wrap the encoder reset function */
void resetEncoder(int i) {
    if (i == LEFT) {
        noInterrupts();
        left_enc_pos = 0L;
        interrupts();
        return;
    } else {
        noInterrupts();
        right_enc_pos = 0L;
        interrupts();
        return;
    }
```

```
}

void resetEncoders() {
    resetEncoder(LEFT);
    resetEncoder(RIGHT);
}
```

馬達接線：

馬達通過 L298N 控制具體接線方式如下：

- L298N 的 ENA 接 Arduino Mega 2560 的 pin5
- L298N 的 ENB 接 Arduino Mega 2560 的 pin6
- IN1(或 A1)和 IN2(或 A2)接 Arduino Mega 2560 的 pin7 和 pin8
- IN3(或 B1)和 IN4(或 B2)接 Arduino Mega 2560 的 pin9 和 pin10

以 IN1 和 IN2 控制馬達 A 轉動方向(正轉或反轉)，ENA 則是控制馬達轉速；

以 IN3 和 IN4 控制馬達 B 轉動方向(正轉或反轉)，ENB 控制馬達的轉速。

`motor_driver.h` 程式碼：

```
void initMotorController();
void setMotorSpeed(int i, int spd);
void setMotorSpeeds(int leftSpeed, int rightSpeed);
```

`motor_driver.ino` 程式碼：

```
#include "commands.h";
// motor A
int enA = 5;
int in1 = 7;
int in2 = 8;
// motor B
int enB = 6;
int in3 = 9;
int in4 = 10;
boolean directionLeft = false;
```

```
boolean directionRight = false;

boolean direction(int i) {
    if (i == LEFT) {
        return directionLeft;
    } else {
        return directionRight;
    }
}

void initMotorController() {
    // set all the motor control pins to outputs
    pinMode(enA, OUTPUT);
    pinMode(enB, OUTPUT);
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(in3, OUTPUT);
    pinMode(in4, OUTPUT);
}

void setMotorSpeed(int i, int spd) {
    if (spd > MAX_PWM) {
        spd = MAX_PWM;
    }
    if (spd < -MAX_PWM) {
        Spd = -1 * MAX_PWM;
    }
    if (i == LEFT) {
        if (spd >= 0) {
            directionLeft = FORWARDS;
            digitalWrite(in2, HIGH);
            digitalWrite(in1, LOW);
            analogWrite(enA, spd);
        } else if (spd < 0) {
            directionLeft = BACKWARDS;
            digitalWrite(in1, HIGH);
            digitalWrite(in2, LOW);
            analogWrite(enA, -spd);
        }
    } else {
        if (spd >= 0) {
            directionRight = FORWARDS;
```

```
            digitalWrite(in3, HIGH);
            digitalWrite(in4, LOW);
            analogWrite(enB, spd);
        } else if (spd < 0) {
            directionRight = BACKWARDS;
            digitalWrite(in4, HIGH);
            digitalWrite(in3, LOW);
            analogWrite(enB, -spd);
        }
    }
}

void setMotorSpeeds(int leftSpeed, int rightSpeed) {
    setMotorSpeed(LEFT, leftSpeed);
    setMotorSpeed(RIGHT, rightSpeed);
}
```

PID 演算法

- 不需要修改 PID 的程式碼，可以直接使用。
- 需要指出的是，它的左右輪的 PID 用的是同一套 `Kp`，`Kd`，`Ki`，`Ko`，如果你的兩個馬達特性差異比較大，就需要對這部分做優化。

`diff_controller.h` 源碼

```
typedef struct {
    double TargetTicksPerFrame;  // target speed in ticks per frame
    long Encoder;   // encoder count
    long PrevEnc;   // last encoder count
    int PrevInput;  // last input
    int ITerm;   // integrated term
    long output; // last motor setting
} SetPointInfo;

SetPointInfo leftPID, rightPID;

int Kp = 20;
int Kd = 12;
int Ki = 0;
```

```
int Ko = 50;

unsigned char moving = 0;  // is the base in motion?

void resetPID() {
    leftPID.TargetTicksPerFrame = 0.0;
    leftPID.Encoder = readEncoder(LEFT);
    leftPID.PrevEnc = leftPID.Encoder;
    leftPID.output = 0;
    leftPID.PrevInput = 0;
    leftPID.ITerm = 0;

    rightPID.TargetTicksPerFrame = 0.0;
    rightPID.Encoder = readEncoder(RIGHT);
    rightPID.PrevEnc = rightPID.Encoder;
    rightPID.output = 0;
    rightPID.PrevInput = 0;
    rightPID.ITerm = 0;
}

void doPID(SetPointInfo*p) {
    long Perror;
    long output;
    int input;
    input = p->Encoder - p->PrevEnc;
    Perror = p->TargetTicksPerFrame - input;

    output = (Kp * Perror - Kd * (input - p->PrevInput) + p->ITerm) /
Ko;
    p->PrevEnc = p->Encoder;

    output += p->output;
    if (output >= MAX_PWM) {
        output = MAX_PWM;
    } else if (output <= -MAX_PWM) {
        output = -MAX_PWM;
    } else {
        p->ITerm += Ki * Perror;
    }

    p->output = output;
    p->PrevInput = input;
```

```
}

void updatePID() {
    leftPID.Encoder = readEncoder(LEFT);
    rightPID.Encoder = readEncoder(RIGHT);

    if (!moving) {
        if (leftPID.PrevInput != 0 || rightPID.PrevInput != 0)
            resetPID();
        return;
    }

    doPID(&rightPID);
    doPID(&leftPID);
    setMotorSpeeds(leftPID.output, rightPID.output);
}

long readPidIn(int i) {
    long pidin = 0;
    if (i == LEFT) {
        pidin = leftPID.PrevInput;
    } else {
        pidin = rightPID.PrevInput;
    }
    return pidin;
}

long readPidOut(int i) {
    long pidout = 0;
    if (i == LEFT) {
        pidout = leftPID.output;
    } else {
        pidout = rightPID.output;
    }
    return pidout;
}
```

第三篇下位機主程式

說明

介紹下位機主程式的工作流程及對應原始碼

工作流程

- 即時讀取上位機的指令並作出對應的回應(支援的指令見原始檔案 `commands.h`)
- `GET_BAUDRATE`：取串口通訊的位元速率
- `READ_ENCODERS`：讀取左右輪編碼器的計數
- `MOTOR_SPEEDS`：設置左右輪的期望速度
- `RESET_ENCODERS`：重置編碼器的計數
- `UPDATE_PID`：更新 PID 參數
- `READ_PIDOUT`：讀取 PID 計算的 PWM 值，為後續調整 PID 參數提供參考
- `READ_PIDIN`：讀取一個 PID 週期內編碼器的計數，為後續調整 PID 參數提供參考

在每個 PID 週期內，比較左右輪期望速度和實際速度的差異，調整 PWM 的。如果

`AUTO_STOP_INTERVAL` 毫秒內上位機沒有到達期望速度，就會停止小車運動。

主程序原始碼

原教程少定義了命令，會導致出錯，此為更正版本。

`ROSArduinoBridge.ino` 程式碼：

```
#include "Arduino.h"
#include "commands.h"
#include "motor_driver.h"
#include "encoder_driver.h"
#include "diff_controller.h"
```

```
#define BAUDRATE      115200
#define MAX_PWM       255

#define PID_RATE      30    // Hz
const int PID_INTERVAL = 1000 / PID_RATE;
unsigned long nextPID = PID_INTERVAL;

#define AUTO_STOP_INTERVAL 2000
long lastMotorCommand = AUTO_STOP_INTERVAL;

/* Variable initialization */
int arg = 0;
int index = 0;
char chr;
char cmd;
char argv1[16];
char argv2[16];
long arg1;
long arg2;

void resetCommand() {
    cmd = NULL;
    memset(argv1, 0, sizeof(argv1));
    memset(argv2, 0, sizeof(argv2));
    arg1 = 0;
    arg2 = 0;
    arg = 0;
    index = 0;
}

int runCommand() {
    int i = 0;
    char* p = argv1;
    char* str;
    int pid_args[4];
    arg1 = atoi(argv1);
    arg2 = atoi(argv2);

    switch (cmd) {
        case GET_BAUDRATE:
            Serial.println(BAUDRATE);
            break;
```

```
case READ_PIDIN:
    Serial.print(readPidIn(LEFT));
    Serial.print(" ");
    Serial.println(readPidIn(RIGHT));
    break;
case READ_PIDOUT:
    Serial.print(readPidOut(LEFT));
    Serial.print(" ");
    Serial.println(readPidOut(RIGHT));
    break;
case READ_ENCODERS:
    Serial.print(readEncoder(LEFT));
    Serial.print(" ");
    Serial.println(readEncoder(RIGHT));
    break;
case RESET_ENCODERS:
    resetEncoders();
    resetPID();
    Serial.println("OK");
    break;
case MOTOR_SPEEDS:
    lastMotorCommand = millis();
    if (arg1 == 0 && arg2 == 0) {
        setMotorSpeeds(0, 0);
        moving = 0;
    } else {
        moving = 1;
    }
    leftPID.TargetTicksPerFrame = arg1;
    rightPID.TargetTicksPerFrame = arg2;
    Serial.println("OK");
    break;
case UPDATE_PID:
    while ((str = strtok_r(p, ":", &p)) != '\0') {
        pid_args[i] = atoi(str);
        i++;
    }
    Kp = pid_args[0];
    Kd = pid_args[1];
    Ki = pid_args[2];
    Ko = pid_args[3];
    Serial.println("OK");
```

```
            break;
        case ANALOG_READ:
            Serial.println(analogRead(arg1));
            break;
        case DIGITAL_READ:
            Serial.println(digitalRead(arg1));
            break;
        case ANALOG_WRITE:
            analogWrite(arg1, arg2);
            Serial.println("OK");
            break;
        case DIGITAL_WRITE:
            if (arg2 == 0) {
                digitalWrite(arg1, LOW);
            } else if (arg2 == 1) {
                digitalWrite(arg1, HIGH);
            }
            Serial.println("OK");
            break;
        case PIN_MODE:
            if (arg2 == 0) {
                pinMode(arg1, INPUT);
            } else if (arg2 == 1) {
                pinMode(arg1, OUTPUT);
            }
            Serial.println("OK");
            break;
        default:
            Serial.println("Invalid Command");
            break;
    }
}

unsigned long time = 0, old_time = 0;

void setup() {
    Serial.begin(BAUDRATE);
    initEncoders();
    initMotorController();
    resetPID();
}
```

```
void loop() {
    while (Serial.available() > 0) {
        chr = Serial.read();
        if (chr == 13) {
            if (arg == 1) {
                argv1[index] = NULL;
            } else if (arg == 2) {
                argv2[index] = NULL;
            }
            runCommand();
            resetCommand();
        } else if (chr == ' ') {
            if (arg == 0) {
                arg = 1;
            } else if (arg == 1) {
                argv1[index] = NULL;
                arg = 2;
                index = 0;
            }
            continue;
        } else {
            if (arg == 0) {
                cmd = chr;
            } else if (arg == 1) {
                argv1[index] = chr;
                index++;
            } else if (arg == 2) {
                argv2[index] = chr;
                index++;
            }
        }
    }

    if (millis() > nextPID) {
        updatePID();
        nextPID += PID_INTERVAL;
    }

    if ((millis() - lastMotorCommand) > AUTO_STOP_INTERVAL) {
        ;
        setMotorSpeeds(0, 0);
        moving = 0;
```

```
    }
}
```

原教程少定義了命令，會導致出錯，此為更正版本。

commands.h 程式碼：

```
#define GET_BAUDRATE    'b'
#define READ_ENCODERS   'e'
#define MOTOR_SPEEDS    'm'
#define RESET_ENCODERS  'r'
#define UPDATE_PID      'u'
#define READ_PIDOUT     'f'
#define READ_PIDIN      'i'

#define PIN_MODE        'c'
#define DIGITAL_READ    'd'
#define ANALOG_READ     'a'
#define DIGITAL_WRITE   'w'
#define ANALOG_WRITE    'x'

#define LEFT            0
#define RIGHT           1
#define FORWARDS        true
#define BACKWARDS       false
```

第四篇上位機程式

說明

介紹上位機樹莓派及相關套裝軟體的安裝及運行

ros_arduino_bridge 安裝

假設上位機已安裝 ROS indigo 版本，並創建了用戶的 ROS 項目空間~/catkin_ws。

程式碼安裝方法：

```
~$ cd ~/catkin_ws/src
~/catkin_ws/src$ git clone https://github.com/hbrobotics/ros_arduino_bridge.git
~/catkin_ws/src$ cd ros_arduino_bridge
(~/catkin_ws/src/ros_arduino_bridge$ git checkout indigo-devel)
~/catkin_ws/src/ros_arduino_bridge$ git checkout kinetic-devel
~/catkin_ws/src/ros_arduino_bridge$ cd ~/catkin_ws
~/catkin_ws$ catkin_make --pkg ros_arduino_bridge
~/catkin_ws$ catkin_make
```

(上述兩筆綠色指令有何差異？二擇一執行)

```
~/catkin_ws$ source devel/setup.bash
```

創建一個設定檔(**car_arduino_params.yaml**)：

~/catkin_ws/src/ros_arduino_bridge/ros_arduino_python/config/car_arduino_params.yaml

car_arduino_params.yaml 原始碼：

```
port: /dev/ttyACM0
baud: 115200
timeout: 0.1
rate: 50
sensorstate_rate: 10

use_base_controller: True
base_controller_rate: 10

# For a robot that uses base_footprint, change base_frame to base_footprint
base_frame: base_link
```

```
# === Robot drivetrain parameters
wheel_diameter: 0.15
wheel_track: 0.287
encoder_resolution: 1768 # from Pololu for 13*34*4 motors
gear_reduction: 1.0
motors_reversed: False

# === PID parameters
Kp: 10
Kd: 12
Ki: 0
Ko: 50
accel_limit: 1.0
```

重要參數含義：

- `wheel_diameter`：小車輪子的直徑
- `wheel_track`：兩個輪子的輪間距
- `encoder_resolution`：編碼器一圈的脈衝計數，如果是 AB 相 13*2*2（13 單圈脈衝數*上下沿*AB 相）。
- `gear_reduction`：減速比，比如 1：30，這是設為 1 或 30
- 也有介紹 `encoder_resolution` 也有設置為 13*2*2*30=1560 和 `gear_reduction` 設為 1，但轉動時候左右輪差異不大
- 本人測試設置 `encoder_resolution`：52 和 `gear_reduction`：30，則是效果很好

安裝 `teleop-twist-keyboard`

通過鍵盤控制套裝軟體，在終端機執行安裝指令：

```
sudo apt-get install ros-melodic-teleop-twist-keyboard
```

運行程式

修改 arduino.launch 檔案，將檔案中的「my_arduino_params.yaml」改為

「car_arduino_params.yaml」。

```
~/catkin_ws/$ vim src/ros_arduino_bridge/ros_arduino_python/launch/arduino.launch
<launch>
	<node name="arduino" pkg="ros_arduino_python" type="arduino_node.py" output="screen" clear_params="true">
		<rosparam file=$(find ros_arduino_python)/config/car_arduino_params.yaml" command="load" />
	</node>
</launch>
```

樹莓派新終端執行：

```
~/catkin_ws$ roslaunch ros_arduino_python arduino.launch
```

樹莓派或工作機執行：

```
~/catkin_ws$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

```
edward@UbuntuVM:~/ros/ws_car$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py

Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5       turn 1.0
```

如果是樹莓派執行需要連接鍵盤

利用鍵盤，根據提示的鍵值，就可以控制小車前後左右移動了

鍵盤說明：

```
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
```

「u」前左轉，「i」前進，「o」前右轉，「j」逆時針轉，「l」順時針轉，「,」後退，「.」後左轉，

「m」後右轉

q/z：增速/減速

鍵值定義：

```
moveBindings = {
'i':(1,0,0,0),
'o':(1,0,0,-1),
'j':(0,0,0,1),
'l':(0,0,0,-1),
'u':(1,0,0,1),
',':(-1,0,0,0),
'.':(-1,0,0,1),
'm':(-1,0,0,-1),
'O':(1,-1,0,0),
'I':(1,0,0,0),
'J':(0,1,0,0),
'L':(0,-1,0,0),
'U':(1,1,0,0),
'<':(-1,0,0,0),
'>':(-1,-1,0,0),
'M':(-1,1,0,0),
't':(0,0,1,0),
'b':(0,0,-1,0),
}

speedBindings={
'q':(1.1,1.1),
'z':(.9,.9),
'w':(1.1,1),
```

```
'x':(.9,1),
'e':(1,1.1),
'c':(1,.9),
}
```

'i':(x,y,z,th)，x 軸線速，y 軸為 0，z 軸為 0，th 角速

moveBindings 大寫字母部分代表全向輪使用

第五篇校準

說明

介紹如果小車速度不穩定，通過校準 PID 來改進

問題：

如果發現小車速度互快互慢，或者走不了直線，那是因為 PID 參數給的不合理造成的。

PID 的目的是通過改變馬達 PWM 值，使馬達實際的轉速基本等於期望的轉速。

如果參數不合理，就會出現實際的轉速和期望的轉速相差很遠。

也就是說，我們沒辦法精准控制小車。

校準方法：

把馬達 PWM 值、期望的轉速和實際的轉速這三者的值用圖表即時地描繪出來。

根據 PWM 值和實際的轉速的運動軌跡，不停地修改 PID 的參數，讓期望的轉速和實際的轉速能在

很短時間內的達到一致。

調節順序，先調 P,再調 I，最後調 D，通常只需要 P 和 I 兩個參數就可以了。

程式更改

馬達 PWM 值、期望的轉速和實際的轉速圖表即時展示的方法

修改檔

```
~/catkin_ws/src/ros_arduino_bridge/ros_arduino_python/src/ros_arduino_python/arduino_driver.py
```

找到 `class Arduino`，在下面添加如下代碼：

```
def get_pidin(self):
    values = self.execute_array('i')
```

```
    if len(values) != 2:
        print "get_pidin count was not 2"
        raise SerialException
        return None
    else:
        return values

def get_pidout(self):
    values = self.execute_array('f')
    if len(values) != 2:
        print "get_pidout count was not 2"
        raise SerialException
        return None
    else:
        return values
```

修改檔

~/catkin_ws/src/ros_arduino_bridge/ros_arduino_python/src/ros_arduino_python/base_controller.py 中有四處需要修改。

在 from tf.broadcaster import TransformBroadcaster 下面增加

```
from std_msgs.msg import Int32
```

self.odomPub = rospy.Publisher('odom' , Odometry) 上面添加如下內容：

```
self.lEncoderPub = rospy.Publisher('Lencoder', Int32)
self.rEncoderPub = rospy.Publisher('Rencoder', Int32)
self.lPidoutPub = rospy.Publisher('Lpidout', Int32)
self.rPidoutPub = rospy.Publisher('Rpidout', Int32)
self.lVelPub = rospy.Publisher('Lvel', Int32)
self.rVelPub = rospy.Publisher('Rvel', Int32)
```

在 poll(self)函數的，if now > self.t_next: 下添加

```
try:
    left_pidin, right_pidin = self.arduino.get_pidin()
except:
    rospy.logerr("getpidout exception count: ")
    return
```

```
self.lEncoderPub.publish(left_pidin)
self.rEncoderPub.publish(right_pidin)
try:
    left_pidout, right_pidout = self.arduino.get_pidout()
except:
    rospy.logerr("getpidout exception count: ")
    return
self.lPidoutPub.publish(left_pidout)
self.rPidoutPub.publish(right_pidout)
```

在 poll(self)函數的，if not self.stopped: 下添加：

```
self.lVelPub.publish(self.v_left)
self.rVelPub.publish(self.v_right)
```

運行程式

在樹莓派終端執行命令：

```
$roslaunch ros_arduino_python arduino.launch
```

在樹莓派新終端執行命令：

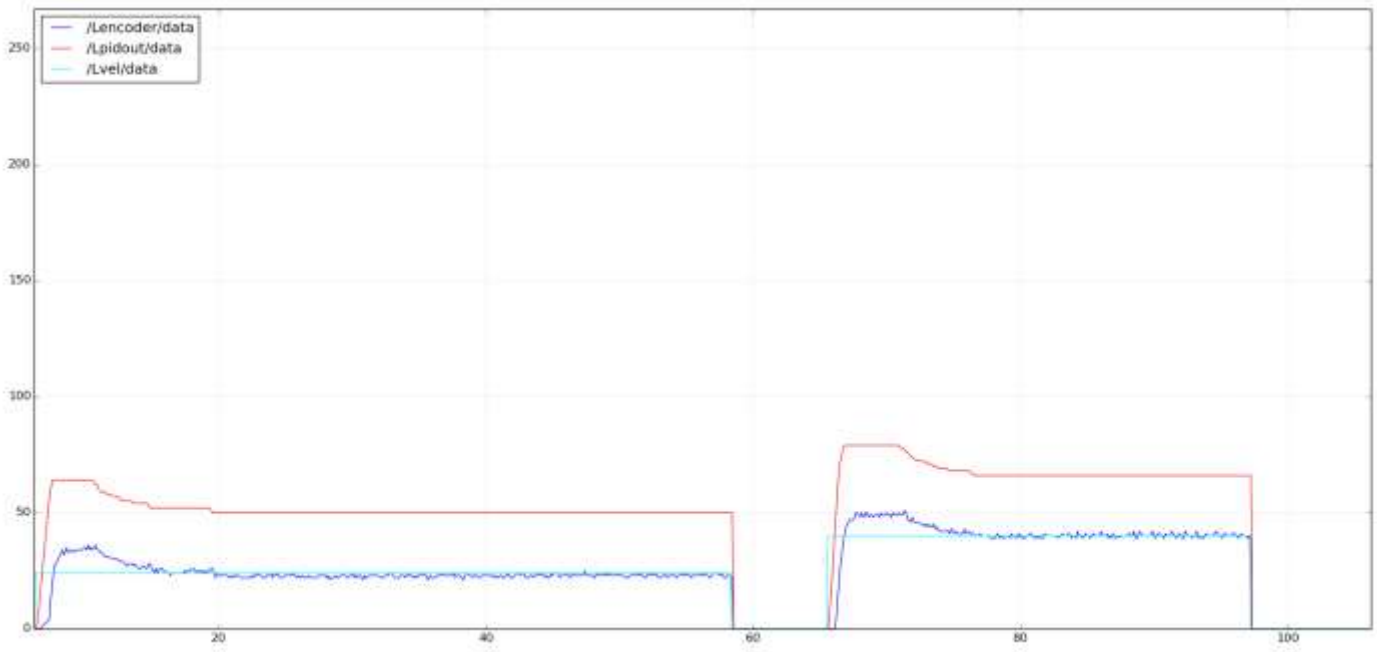
```
$rqt_plot /Lencoder /Lpidout /Lvel
```

在樹莓派新終端執行命令：

```
$rostopic pub -r 30  /cmd_vel geometry_msgs/Twist '{linear: {x: 0.3, y:
0, z: 0}, angular: {x: 0, y: 0, z: 0}}'
```

小車以 0.3m/s 固定線速度運行。圖表中/Lvel/data 代表左輪期望轉速 (以編碼器的計數器數目表示)，/Lencoder/data 代表左輪實際的轉速，/Lpidout/data 代表馬達 PWM 值。右輪的三個值分別是/Rvel、/Rencoder 和/Rpidout。

效果圖：



我們可以改變不同的速度，調整並檢查 PID 效果。

PID 調節口訣

口訣 1：

參數整定尋最佳，從大到小順次查。

先是比例後積分，最後再把微分加。

曲線振盪很頻繁，比例度盤要放大。

曲線漂浮繞大彎，比例度盤往小扳。

曲線偏離回復慢，積分時間往下降。

曲線波動週期長，積分時間再加長。

理想曲線兩個波，調節過程高品質。

口訣 2：

參數整定找最佳，從小到大順序查

先是比例後積分，最後再把微分加

曲線振盪很頻繁，比例度盤要放大

曲線漂浮繞大灣，比例度盤往小扳

曲線偏離回復慢，積分時間往下降

曲線波動週期長，積分時間再加長

曲線振盪頻率快，先把微分降下來

動差大來波動慢。微分時間應加長

理想曲線兩個波，前高後低 4 比 1

一看二調多分析，調節品質不會低。

第六篇 ros_arduino_bridge 功能包的使用

說明

介紹 `ros_arduino_bridge` 功能包

ROSArduinoBridge

這個 ROS 功能包集包括了 Arduino 庫（ROSArduinoBridge）和一系列用來控制基於 Arduino 的 ROS 功能包，它使用的是標準的 ROS 消息和服務。這個功能包集並不依賴於 ROS 串口

使用這個功能包時，注意選擇相應的版本，因為不同的版本之間有些並不能相容。indigo-devel branch 是針對 ROS Indigo 以及更高版本的，它使用的是 Catkin 編譯系統。

這個功能包集包括一個相容不同驅動的機器人的基本控制器（base controller），它可以接收 ROS Twist 類型的消息，可以發佈里程資料到個人電腦。這個控制器（base controller）要求使用一個馬達控制器和編碼器來讀取里程資料

特點

可以直接支援 ping 聲吶和 Sharp 紅外線感測器

也可以從通用的類比和數位信號的感測器讀取資料

可以控制數字信號的輸出

支持 PWM 伺服機

如果使用所要求的硬體的話，可以配置基本的控制

如果你的 Arduino 程式設計基礎好的話，並且具有 python 基礎的話，你就可以很自由的改動代碼來滿足你的硬體要求

目前 indigo-devel 版本的功能包集支援下面的控制器硬體

Pololu VNH5019 dual motor controller shield or Pololu MC33926 dual motor shield.
Robogaia Mega Encoder shield or on-board wheel encoder counters.

注意：

Robogaia Mega Encoder shield 僅適用於 Arduino Mega，

板上編碼計數器(ARDUINO_ENC_COUNTER)目前僅支持 Arduino Uno

上面非硬性規定，有一定的程式設計基礎，你也可以按需更改

系統要求：

1.安裝 python-serial 功能包（Ubuntu）

介紹：這個功能包集可以在相容 Arduino 的控制器上進行讀取感測器上的資料，以及控制 PWM 伺服機。但是你必須具備上面所說的被支援的硬體（電動機控制器和編碼器），你才能使用這個功能包集中的基本控制器（base controller）

安裝：

```
$ sudo apt-get install python-serial
```

或

```
$ sudo pip install --upgrade pyserial
```

或

```
$ sudo easy_install -U pyserial
```

2.安裝電動機控制器和編碼器合適的庫

對於上面提到過的 Pololu VNH5019 Dual Motor Shield，它的相應的庫可以在這裡找到：訪問庫

對於 Pololu MC33926 Dual Motor Shield，則在這裡可以找到對應的庫：訪問庫

Robogaia Mega Encoder shield 的庫可以在這裡找到:訪問庫

這些庫應該被安裝到你的標準 Arduino sketchbook/libraries 路徑下面

這個功能包集假設你使用的 Arduino IDE 的版本是 1.0 或以上。

Linux 下連接 Arduino

1. Arduino 很可能是通過介面/dev/ttyACM#或者/dev/ttyUSB#來連接你的 Linux 系統的。這裡的#可以是 0、1、2 等數，當然這根據你連接的設備數量而定。得到這個數位#最簡單的方式就是拔掉所有的 USB 設備，然後插上你的 Arduino，然後運行下面這個命令

```
~$ ls /dev/ttyACM*
```

或

```
~$ ls /dev/ttyUSB*
```

2. 上面這兩個命令的有一個可以返回你想要的結果（例如/dev/ttyACM0），假設你的 Arduino 連接的是/dev/ttyACM0

3. 查看介面的存取權限

```
~$ ls -l /dev/ttyACM0
crw-rw-- 1 root dialout 166, 0 2013-02-24 08:31 /dev/ttyACM0
```

上面的結果中，只有 root 和"dialout"群組才有讀寫許可權，你需要成為 dialout 群組的一個成員。你僅僅需要馬上加入這組中，並且它會對你之後插入的所有的 USB 設備都起作用。

4. 加入 dialout 群組的指令

```
~$ sudo usermod -a -G dialout your_user_name
```

命令中 your_user_name 就是你在 Linux 下登錄的用戶名。之後你可能需要登出，然後再重新登錄進去，或者就簡單的重啟一下電腦。確認已經成功讓用戶加入組，如果你可以在列出的群組中找到 dialout，這就說明你已經加入到 dialout 中了

```
~$ groups
```

安裝 ros_arduino_bridge 功能包集

1. 安裝 ros_arduino_bridge

```
~$ cd ~/catkin_ws/src
~/catkin_ws/src$ git clone
https://github.com/hbrobotics/ros_arduino_bridge.git
~/catkin_ws/src$ cd ~/catkin_ws
~/catkin_ws$ catkin_make
```

這個被提供的 Arduino 庫叫做 ROSArduinoBridge，你可以在 ros_arduino_firmware 功能包中找到

2. 安裝 ROSArduinoBridge 庫：

進入 SKETCHBOOK_PATH 是指你的 Arduino sketchbook 路徑

```
~$ cd SKETCHBOOK_PATH
```

cp 命令就是把 ROSArduinoBridge sketch 檔拷貝到你的 sketchbook 目錄下面

```
$ cp -rp `rospack find
ros_arduino_firmware`/src/libraries/ROSArduinoBridge ROSArduinoBridge
```

如果之前已經有先刪除

```
$ cd SKETCHBOOK_PATH
$ rm -R ROSArduinoBridge
```

3.編譯上傳 ROSArduinoBridge 的 Sketch

打開 Arduino IDE 找到 File->Sketchbook->ROSArduinoBridge

如果你正在使用基礎控制器功能包（base controller），那麼你必須確保你已經在 Arduino sketchbook/libraries 目錄裡面安裝了合適的電動機控制器和編碼器的相關程式庫。你可以通過去掉或保留相關的巨集定義聲明，來選擇你想用的電動機控制器，同時你還要將其他電動機控制器的宏聲明注釋掉，預設選擇的是 Pololu VNH5019 driver。

你也可以用相同的方法選擇你想使用的編碼器。預設選擇的是 Pololu VNH5019 driver

想讓你的基礎控制器控制 PWM 伺服機的話，找到程式碼：

```
//#define USE_SERVOS
#undef USE_SERVOS
```

更改為：

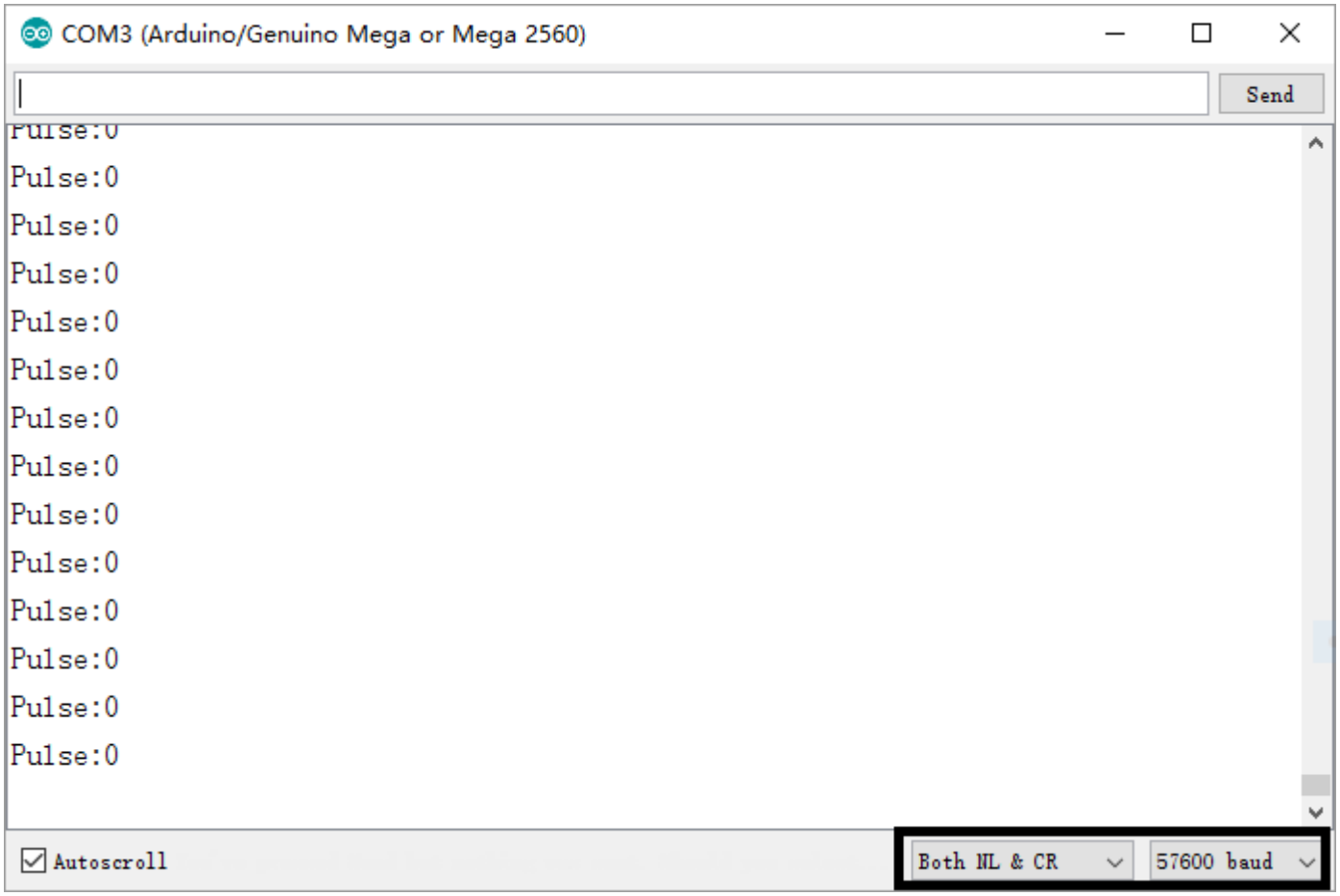
```
#define USE_SERVOS
//#undef USE_SERVOS
```

修改標頭檔 servos.h 改變其中的 N_SERVOS 參數，你還需要根據你的伺服機所接的引腳修改相應的引腳數字。一切都準備好後，你就可以把這個 sketch 編譯並上傳到你的 Arduino 控制板。

固件測試

1. 串口監視器

這個 ROSArduino 程式庫利用單一字母來輪詢感測器、控制伺服機、驅動機器人以及讀取編碼器。這些命令可以通過序列埠介面來發送給 Arduino，比如 Arduino 的序列監視器，把序列監視器的序列傳輸速率設置為 57600 然後把行結束符設置為"Carriage return"和"Both NL & CR"。這些設置選項在序列監視器右下角的兩個下拉式功能表中。如下圖所示：



2. 目前的列表包括這些命令(命令列表可以在標頭檔 commands.h 中找到)

```
#define ANALOG_READ 'a'
#define GET_BAUDRATE   'b'
#define PIN_MODE 'c'
#define DIGITAL_READ   'd'
#define READ_ENCODERS  'e'
#define MOTOR_SPEEDS   'm'
#define PING 'p'
#define RESET_ENCODERS  'r'
#define SERVO_WRITE 's'
#define SERVO_READ  't'
#define UPDATE_PID  'u'
#define DIGITAL_WRITE  'w'
#define ANALOG_WRITE   'x'
```

3. 命令使用例子

如果你想從模擬引腳 pin 3 讀取資料，就發送這個命令：a 3

你想改變數位引腳 pin 3 的模式為 OUTPUT，就發送這個命令：c 3 1

得到目前編碼器的計數，就發送這個命令：e

讓機器人以每秒 20 個 encoder ticks 的速度向前移動，就發送這個命令：m 20 20

這些命令有些需要有對應的參數，比如 a 3 中 3 代表模擬引腳 pin 3

連線測試

在一個差速輪式機器人上，電動機使用兩個極性相反的接頭來連接到電動機控制器上的，同樣地，

連接到編碼器上的 A/B 端也是互斥的。然而，你仍然需要確保這樣兩個要求：

- 當給一個正向的速度，輪子向前移動；
- 而當輪子向前移動時，編碼器計數增加。

把機器人的輪子架空，你可以使用串口監視器來測試上述的兩個要求。

你可以使用 m 命令來啟動電動機，使用 e 命令來獲取編碼器計數，以及使用 r 命令來將編碼器重置為 0。

在固件層，電動機的速度是按照編碼器每秒的 tick 數來表示的，這樣可以使編碼器很容易解析 (理解) 它。

假如輪子轉速是每秒 4000 個編碼器計數，那麼命令 m 20 20 會以一個非常小的速度移動小車。(默認設置輪子僅僅移動 2 秒，你可以通過修改源碼中的 AUTO_STOP_INTERVAL 改變這個值)，第一個參數是左輪的速度，另一個參數是右輪的速度。

當使用 e 命令時，第一個被返回的數是左輪的編碼器計數，第二個數是右輪的編碼器計數。

可以先使用 r 命令把編碼器計數清零，然後通過手動粗略地旋轉輪子一整圈，再通過 e 命令來驗證獲取到的編碼器計數是否為預期的結果。

配置 ros_arduino_python 節點

1. 編輯 ros_arduino_python/config 路徑下的 YAML 檔定義機器人的尺寸，PID 參數，以及感測器配置資訊。

```
$ roscd ros_arduino_python/config
$ cp arduino_params.yaml my_arduino_params.yaml #複製一個副本
$ vim my_arduino_params.yaml #查看內容
```

2. my_arduino_params.yaml 代碼

```
# For a direct USB cable connection, the port name is typically
# /dev/ttyACM# where # is a number such as 0, 1, 2, etc
```

```
# For a wireless connection like XBee, the port is typically
# /dev/ttyUSB# where # is a number such as 0, 1, 2, etc.

port: /dev/ttyACM0
baud: 57600
timeout: 0.1

rate: 50
sensorstate_rate: 10

use_base_controller: False
base_controller_rate: 10

# For a robot that uses base_footprint, change base_frame to
base_footprint
base_frame: base_link

# === Robot drivetrain parameters
#wheel_diameter: 0.146
#wheel_track: 0.2969
#encoder_resolution: 8384 # from Pololu for 131:1 motors
#gear_reduction: 1.0
#motors_reversed: True

# === PID parameters
#Kp: 10
#Kd: 12
#Ki: 0
#Ko: 50
#accel_limit: 1.0

# === Sensor definitions.  Examples only - edit for your robot.
#     Sensor type can be one of the following:
#     * Ping
#     * GP2D12
#     * Analog
#     * Digital
#     * PololuMotorCurrent
#     * PhidgetsVoltage
#     * PhidgetsCurrent (20 Amp, DC)

sensors: {
```

```
  #motor_current_left:  {pin: 0, type: PololuMotorCurrent, rate: 5},
  #motor_current_right: {pin: 1, type: PololuMotorCurrent, rate: 5},
  #ir_front_center: {pin: 2, type: GP2D12, rate: 10},
  #sonar_front_center:  {pin: 5, type: Ping, rate: 10},
  onboard_led:   {pin: 13, type: Digital, rate: 5, direction: output}
}

# Joint name and configuration is an example only
joints: {
   head_pan_joint:  {pin: 3, init_position: 0, init_speed: 90, neutral:
90, min_position: -90, max_position: 90, invert: False, continuous:
False},
   head_tilt_joint: {pin: 5, init_position: 0, init_speed: 90, neutral:
90, min_position: -90, max_position: 90, invert: False, continuous:
False}
}
```

不要在你的.yaml 檔裡使用 tab，否則的話這個語法解析器會無法載入它。如果需要縮排的話必須用空格代替 tab 鍵。當定義你的感測器參數時，列表中的最後一個感測器的行尾（大括弧後面）後沒有逗號，但是其餘的行尾必須有逗號。

3. 解釋代碼

- 介面設置

介面要麼是/dev/ttyACM0，要麼是/dev/ttyUSB0，視情況而定

其中 MegaRobogaiaPololu 的 Arudino sketch 默認是以 57600 的串列傳輸速率連接的。

- 輪詢速率

跳出 ROS loop 運行的速率主要就取決於這個速率參數（預設為 50Hz），這個預設值足以滿足大多數情況。在任何情況下，它應該至少與你的感測器的最大速率（下面我們會說到）一樣快才行

sensorstate_rate 決定了多久發佈一個所有感測器的集合清單，每個感測器也以各自的速率在各自的主題上發佈消息。

use_base_controller 參數默認為 False。你可以把它設置為 True（假設你有文中所要求的硬體設施）。

設置 PID 參數 base_controller_rate 參數決定了多久發佈一次里程計讀取資訊。

- 定義感測器

sensors 參數按照定義了感測器的名字和參數的字典（可任意指定，感測器的名字也會成為那個感測器所對應主題的名字）

四個最重要的參數分別是：pin，type，rate，direction。

rate 定義了你每秒想輪詢一次那個感測器多少次。例如，一個電壓感測器可能每秒僅僅被輪詢一次（或者僅僅每兩秒一次），而一個聲吶感測器可能每秒被輪詢 20 次。

type 必須是列表中被列出來的（注意區分大小寫！）。

direction 默認是 input，所以如果你想將它定義為 output，就將這個 direction 單獨設為 output。

在上面的例子中，Arduino LED（pin13）將會以每秒兩次的速率被點亮或熄滅。

- 設置 Drivetrain（驅動系統）和 PID 參數

為了使用基礎控制器（base controller），你必須去掉它的注釋並且設置機器人的 drivetrain 和 PID 參數

示例中 drivetrain 參數是直徑 6 英寸的驅動輪，距離 11.5 英寸

注意在 ROS 中使用米作為距離單位，所以一定要換算單位

示例中的編碼器的解析度（每轉的 tick 數）規格來自於 Pololu 131:1 電動機。為你的電動機/編碼器組合設置合理的數值

如果你的輪子可以向後轉，那麼就把 motors_reversed 設置為 true，否則的話就設置為 False

PID 參數比較難設置，你可以先按照示例中的值設置。但是在你第一次發送轉彎命令的時候，架空

小車測試。

啟動 ros_arduino_python 節點

1.查看 ros_arduino_python/launch 下的文件 arduino.launch，它指向一個名叫

my_arduino_params.yaml 的文件

```
<launch>
  <node name="arduino" pkg="ros_arduino_python" type="arduino_node.py"
output="screen" clear_params="true">
    <rosparam file="$(find
ros_arduino_python)/config/my_arduino_params.yaml" command="load" />
  </node>
</launch>
```

如果你的設定檔命名不同，那麼就把這裡的檔案名參數(my_arduino_params.yaml)修改成你的

設定檔的名字

2.連接好 Arduino，關閉串口監視器，啟動 ros_arduino_python node 節點

```
$ roslaunch ros_arduino_python arduino.launch
```

啟動之前，千萬不要打開 Arduino IDE 的串口監視器，因為串口監視器會與該節點爭奪串口資源。

出現效果：

```
process[arduino-1]: started with pid [6098]
Connecting to Arduino on port /dev/ttyUSB0 …
Connected at 57600
Arduino is ready.
[INFO] [WallTime: 1355498525.954491] Connected to Arduino on port / >
dev/ttyUSB0 at 57600 baud
[INFO] [WallTime: 1355498525.966825] motor_current_right {'rate': 5, >
'type': 'PololuMotorCurrent', 'pin': 1}
[INFO]
```

```
etc
```

如果你在你的機器人上裝的有 Ping 聲吶而且你也在設定檔裡面定義了它們，它們就會閃一下來告訴你已經連接成功

查看感測器資料

1.查看所有感測器的資料

```
$ rostopic echo /arduino/sensor_state
```

2.查看任何指定的感測器資料

```
$ rostopic echo /arduino/sensor/sensor_name
```

例如：有一個叫做 `ir_front_center` 的感測器，查看相應的資料

```
$ rostopic echo /arduino/sensor/ir_front_center
```

3.使用 rxqrt 來將這系列資料用圖像的形式表示出來

```
$ rxplot -p 60 /arduino/sensor/ir_front_center/range
```

發送 Twist 命令與查看里程計數據

1.把你的機器人放到塊上使輪子懸空，然後發佈一個 Twist 命令

```
$ rostopic pub -1 /cmd_vel geometry_msgs/Twist '{ angular: {z: 0.5} }'
```

兩個輪子的轉動方向應該是一致的，都是逆時針轉動(右輪前進，左輪後退)。如果它們轉動方向相反，那麼就將 `motors_reversed` 參數改為與之前相反的值，然後用 `ctrl+c` 停止該節點，然後重新啟動 `arduino.launch` 文件。

2.使用下面的命令可以使機器人停止

```
$ rostopic pub -1 /cmd_vel geometry_msgs/Twist '{}'
```

3.查看里程資料可以運行

```
$ rostopic echo /odom
```

或者使用圖形介面顯示：

```
$ rxplot -p 60 /odom/pose/pose/position/x:y,
/odom/twist/twist/linear/x, /odom/twist/twist/angular/z
```

ROS 服務

1.digital_set_direction-設置數字引腳的方向

```
$ rosservice call /arduino/digital_set_direction pin direction
```

這裡 pin 是引腳數字，direction 為 0 代表輸入，1 代表輸出。

2.digital_write-給數字引腳發送高低電平（LOW 為 0，HIGH 為 1）

```
$ rosservice call /arduino/digital_write pin value
```

同樣，這裡 pin 是引腳數字，value 是電平高低（LOW 為 0，HIGH 為 1）。

3.servo_write-設置伺服機位置

```
$ rosservice call /arduino/servo_write id pos
```

這裡 id 是伺服機的索引號（定義在 Arduino sketch 中的 servos.h）並且 pos 是以弧度為單位

（0-3.14），標頭檔 servos.h 中具體是這樣寫的：

```
byte servoInitPosition [N_SERVOS] = { 90, 90 }; // [0, 180] degrees
```

4.servo_read-讀取伺服機的位置

```
$ rosservice call /arduino/servo_read id
```

使用板上編碼器計數（僅支援 Arduino Uno）

對於 Arduino Uno，這個固件程式支援板上的編碼器計數。這樣的話，編碼器就直接可以連接到

Arduino 板上，而不用借助任何額外的編碼器設備（例如 RoboGaia encoder shield）

對於速度，這個代碼可以直接找到 Atmega328p 的介面和中斷管腳，而這一功能的實現必須依賴

Atmega328p（Arduino Uno）。（儘管它也可能相容其他電路板或 AVR 單片機晶片）

為了使用這個板上編碼器計數，按照下面的要求來將編碼器連接到 Arduino Uno：

```
Left wheel encoder A output -- Arduino UNO pin 2
Left wheel encoder B output -- Arduino UNO pin 3

Right wheel encoder A output -- Arduino UNO pin A4
Right wheel encoder B output -- Arduino UNO pin A5
```

這時你需要在 Arduino sketch 中做相應的修改，通過下面的方式來取消使用 RoboGaia encoder shield 的代碼，啟用板上編碼器的代碼。

```
/* The RoboGaia encoder shield */
//#define ROBOGAIA
/* Encoders directly attached to Arduino board */
#define ARDUINO_ENC_COUNTER
```

這時你就可以編譯並上傳到 Arduino 上了。

補充

1.如果你沒有文檔中所要求的硬體來運行這個基礎控制器，但是你仍然想使用其他相容 Arduino 的控制器來讀取感測器以及控制 PWM 伺服機，那麼請按照下面的步驟：

首先，你需要編輯你的 ROSArduinoBridge sketch，在檔的最前面，將這兩行：

```
#define USE_BASE
//#undef USE_BASE
```

改成這樣:

```
//#define USE_BASE
#undef USE_BASE
```

你還需要將檔 `encoder_driver.ino` 中的這一行注釋掉：

```
#include "MegaEncoderCounter.h"
```

你就可以編譯並上傳你的代碼

然後，編輯你的 my_arduino_params.yaml 文件，確保參數 use_base_controller 被設為 False。這樣你就完成了

參考：

- https://github.com/hbrobotics/ros_arduino_bridge
- 機械工業出版社，ROS 機器人程式設計
- http://answers.ros.org/question/233848/polling-rates/
- http://blog.csdn.net/github_30605157/article/details/51344150

第七篇 ros_arduino_bridge 錯誤及解決

說明：

介紹 ros_arduino_bridge 可能的相關錯誤及解決

1.未能啟動 arduino_node 節點

問題：在你使用 roslaunch ros_arduino_python arduino.launch 啟動該節點之後，可能會遇到下面的錯誤：

```
ERROR: cannot launch node of type [ros_arduino_python/arduino_node.py]:
can't locate node [arduino_node.py] in package [ros_arduino_python].
```

原因及解決：產生這個錯誤的主要原因有兩種：

- 你的電腦並未連接 Arduino 板，或者沒有連接好，這時你就需要檢查一下連接了；
- 如果連接完好，那麼很可能就是你對檔 arduino_node.py 沒有可執行的許可權，所以只要使用命令 chomd a+x arduino_node.py 給它加上可執行許可權就行了.

2.未能確定 Arduino 的串列傳輸速率

問題：

```
Connecting to Arduino on port /dev/ttyACM0 …
Failed to determine baud rate of Arduino so aborting!
```

分析：

- 這個問題其實比較棘手。其實在啟動之前你應該使用串口監視器對這個 Arduino sketch 進行測試的，如果沒有問題的話在進行下一步的配置。筆者曾使用默認的 baud rate(57600)，但是並不能從串口監視器讀出正確的值，甚至什麼也輸不出來。我曾今嘗試過修改代碼中默認的串列傳輸速率，但是代碼的原作者卻堅持認為不應該修改它，但是我將其修改成 9600 卻成功地從串口監視器得到了預期的值。

- 可是我再次運行這個節點的時候，還是會出現這樣的錯誤。最後該功能包的原作者給了我一個這樣的答案：你用的那個分支（master）還在開發中，可能會存在問題，你可以嘗試使用 indigo-devel。其實這個答案讓我很難接受，他並沒有告訴我具體什麼原因導致的這樣的問題，而是以這樣的一個理由來搪塞我。更好況，在 github 的慣例中，master 分支應該是一個隨時可以使用的穩定版，倉庫管理者應該即時合併分支。不過，我也只好按照它的建議使用了另一個分支。果然，換了一個分支之後就不會出現這樣的錯誤了。

解決方法：

- 首先測試 Arduino sketch（即功能包裡的 ros_arduino_firmware 資料夾裡的代碼）是否運行正常；
- 如果仍然出現這樣的錯誤，那麼就請檢查你是用的那個分支是否與你的板子或電腦相容

3.命令執行錯誤

錯誤：

```
Connecting to Arduino on port /dev/ttyACM0 …
Serial Exception:
(<\class 'serial.serialutil.SerialException'>, SerialException(),
<\traceback object at 0x7fbc064c7c20)>
Traceback follows:
Traceback (most recent call last):
File
"/home/bird/catkin_ws/src/ros_ab_indigo/ros_arduino_python/src/ros_ardu
ino_python/arduino_driver.py", line 75, in connect
raise SerialException
SerialException
Cannot connect to Arduino!
[arduino-1] process has died [pid 2849, exit code 1, cmd
/home/bird/catkin_ws/src/ros_ab_indigo/ros_arduino_python/nodes/arduino
_node.py __name:=arduino __log:=/home/bird/.ros/log/3945ca82-2e1e-11e6-
abd3-00e04c7df3a3/arduino-1.log].
```

```
log file: /home/bird/.ros/log/3945ca82-2e1e-11e6-abd3-
00e04c7df3a3/arduino-1*.log
all processes on machine have died, roslaunch will exit
shutting down processing monitor…

… shutting down processing monitor complete

done
```

- 如果你得到這樣的錯誤，很可能就是因為從 serial 獲取的輸出結果有問題。你在 Python 指令檔的相應位置輸出獲取的結果，你就會發現結果是錯誤的。但是究竟是什麼原因導致這樣的問題呢？直接原因就是串口的實際串列傳輸速率與你所設置的串列傳輸速率大小不一致。
- 這時你就需要檢查一下你給 arduino 所設置的串列傳輸速率是否和你的設定檔(my_arduino_params.yaml)中的串列傳輸速率一致。筆者曾近犯過這樣一個愚蠢的錯誤：arduino 板已經被其他人用過，而我忘記了重新向 arduino 上傳 ros_arduino_firmware 中的代碼，結果就出現了上面的錯誤。
- 另外還有一個比較隱晦的原因，就是你同時啟動了 Arduino IDE 的串口監視器，導致它和這個節點爭奪 serial，於是就造成了錯誤或者不完整的資訊。當然，類似多個程式使用同一個串口的原因也會導致這樣的錯誤。

參考：

- https://github.com/hbrobotics/ros_arduino_bridge/issues/27
- http://blog.csdn.net/github_30605157/article/details/51621965

第八篇 ros_arduino_bridge 代碼解讀

說明

介紹 ros_arduino_bridge 套裝軟體檔及相應功能

文件樹及說明：

```
├── README.md
├── ros_arduino_bridge                    # metapackage (元包)
│   ├── CMakeLists.txt
│   └── package.xml
├── ros_arduino_firmware                  #固件包，更新到 Arduino
│   ├── CMakeLists.txt
│   ├── package.xml
│   └── src
│       └── libraries                     #庫目錄
│           ├── MegaRobogaiaPololu  #針對 Pololu 馬達控制器，MegaRobogaia 編碼器的標頭檔定義
│           │   ├── commands.h          #定義命令標頭檔
│           │   ├── diff_controller.h     #差分輪 PID 控制標頭檔
│           │   ├── MegaRobogaiaPololu.ino  #PID 實現檔
│           │   ├── sensors.h             #感測器相關實現，超聲波測距，Ping 函數
│           │   └── servos.h              #伺服器標頭檔
```

```
|           └── ROSArduinoBridge          #Arduino 相關庫定義
|               ├── commands.h            #定義命令
|               ├── diff_controller.h     #差分輪 PID 控制標頭檔
|               ├── encoder_driver.h      #編碼器驅動標頭檔，定義插腳(pins)
|               ├── encoder_driver.ino    #編碼器驅動實現，讀取編碼器資料，重置編碼器等
|               ├── motor_driver.h        #馬達驅動標頭檔
|               ├── motor_driver.ino      #馬達驅動實現，初始化控制器，設置速度
|               ├── ROSArduinoBridge.ino  #核心功能實現，
|               ├── sensors.h             #感測器標頭檔及實現
|             ├── servos.h                #伺服器標頭檔，定義插腳，類
|               └── servos.ino            #伺服器實現
├── ros_arduino_msgs                      #消息定義包
|   ├── CMakeLists.txt
|   ├── msg                               #定義消息
|   |   ├── AnalogFloat.msg               #定義類比 IO 浮點消息
|   |   ├── Analog.msg                    #定義類比 IO 數位消息
|   |   ├── ArduinoConstants.msg          #定義常量消息
|   |   ├── Digital.msg                   #定義數位 IO 消息
|   |   └── SensorState.msg               #定義感測器狀態訊息
|   ├── package.xml
|   └── srv                               #定義服務
```

```
|       ├── AnalogRead.srv            #模擬 IO 輸入
|       ├── AnalogWrite.srv           #模擬 IO 輸出
|       ├── DigitalRead.srv           #數位 IO 輸入
|       ├── DigitalSetDirection.srv   #數字 IO 設置方向
|       ├── DigitalWrite.srv          #數位 IO 輸入
|       ├── ServoRead.srv             #伺服馬達輸入
|       └── ServoWrite.srv            #伺服馬達輸出
└── ros_arduino_python                #ROS 相關的 Python 包，用於上位機，樹莓派等開發板或電腦等。
    ├── CMakeLists.txt
    ├── config                        #配置目錄
    |   └── arduino_params.yaml       #定義相關參數，埠，rate，PID，sensors 等默認參數。由 arduino.launch 調用
    ├── launch
    |   └── arduino.launch            #開機檔案
    ├── nodes
    |   └── arduino_node.py           #python 文件，實際處理節點，由 arduino.launch 調用，可單獨調用
    ├── package.xml
    ├── setup.py
    └── src                           #Python 類包目錄
```

```
        └── ros_arduino_python
            ├── arduino_driver.py          #Arduino 驅動類
            ├── arduino_sensors.py         #Arduino 感測器類
            ├── base_controller.py         #基本控制類，訂閱 cmd_vel 話題，發佈 odom
話題
            └── __init__.py                #類包默認空文件
```

重要知識點：

- metapackage
- Python 類包
- PID 控制原理及實現

第九篇 ros_arduino_python 介紹

說明：

介紹 `ros_arduino_python` 包相關功能

這個包包括一個 Python 驅動程式和 ROS 節點，用於 Arduino 相容的控制器。Arduino 必須使用 USB 埠或 RF 串列鏈路（例如 XBee）連接到 PC 或 SBC。

特徵

直接支援以下感測器：

- Ping sonar
- Sharp infrared (GP2D12)夏普紅外（GP2D12）
- Onboard Pololu motor controller current 板載 Pololu 馬達控制器電流
- Phidgets Voltage sensorPhidgets 電壓感測器
- Phidgets Current sensor (DC 20A)，Phidgets 電流感測器（DC 20A）

還可以從通用類比和數位感測器讀取資料

可以控制數字輸出（例如打開和關閉開關或 LED）

支持 PWM 伺服

如果使用所需的硬體，可配置基本控制器。

Arduino 節點

`arduino-node.py`

- 用於 Arduino 相容微控制器的 ROS 節點。
- 感測器可以以獨立的速率輪詢（參見本頁末尾的示例設定檔）。
- 例如，主電壓可以以 1Hz 的頻率輪詢，而聲納感測器可以以 20Hz 的速度輪詢。

訂閱主題

- /cmd_vel (geometry_msgs/Twist)，基本控制器的移動命令

發佈主題

- /odom (nav_msgs/Odometry)，來自基本控制器的 Odometry 消息。
- ~sensor_state (ros_arduino_msgs/SensorState)，感測器名稱和值的陣列。
- ~sensor/name (sensor_msgs/Range,)，每個感測器值都在相應類型的“sensor”命名空間下的自己的主題上發佈。

服務

- ~servo_write (ros_arduino_msgs/ServoWrite),將索引為'id'的伺服器的目標位置設置為'value'（以弧度表示）。'id'到引腳映射是在 Arduino 固件中進行的。
- ~servo_read (ros_arduino_msgs/ServoRead),從具有索引“id”的伺服獲取最後設置的位置（以弧度表示）。
- ~digital_set_direction (ros_arduino_msgs/DigitalSetDirection),將數字引腳設置為 INPUT（0）或 OUTPUT（1）
- ~digital_write (ros_arduino_msgs/DigitalWrite),將數字引腳設置為低（0）或高（1）

參數

- `~port` (`str`, default: `/dev/ttyUSB0` -- some controllers use `/dev/ttyACM0`)，序列埠
- `~baud` (`int`, default: `57600`),串列連接的串列傳輸速率。
- `~timeout` (`float`, default: `0.1`),串口超時（秒）
- `~rate` (`int`, default: `50`),速率運行主控制迴圈。應至少與更快的感測器速率一樣快。
- `~sensorstate_rate` (`int`, default: `10`),發佈感測器資料的頻率。各個感測器是按照自己

的話題和自己的速度發佈的。

- ~use_base_controller (bool, default: False)，是否使用基本控制器。
- ~base_controller_rate (int, default: 10)，發佈 odometry 資料的頻率
- ~base_frame (New in Hydro) (string, default: base_link)，接受 odometry 數據的坐標系
- ~wheel_diameter (float, default: none)，車輪直徑 (米)
- ~wheel_track (float, default: none)，輪距 (米)。(驅動輪中心之間的距離)
- ~encoder_resolution (int, default: none)，每圈的編碼器解析度。
- ~gear_reduction (float, default: 1.0)，減速比
- ~motors_reversed (bool, default: False)，反轉車輪旋轉方向
- ~Kp (int, default: 20)，PID 參數：比例
- ~Kd (int, default: 12)，PID 參數：微分
- ~Ki (int, default: 0)，PID 參數：積分
- ~Ko (int, default: 50)，PID 參數：輸出
- ~accel_limit (float, default: 0.1)，輪速變化最大的加速度
- ~sensors (dict, default: None)，附加到 Arduino 的感測器字典

提供 tf 轉換

- odom → base_link ， 導航需要的變換

配置

- Arduino 節點使用指定所需參數的 YAML 檔進行配置。
- 名為 arduino_params.yaml 的示例參數檔包含在 config 目錄中，如下所示。

- 在編輯之前製作此檔的副本（例如 my_arduino_params.yaml）。
- 注意，許多參數被注釋掉，必須設置和取消注釋，然後才能將該節點與 Arduino 配合使用。
- 當前有效的感測器類型名稱（區分大小寫）：
  - Ping
  - GP2D12
  - Analog (generic)
  - Digital (generic)
  - PololuMotorCurrent
  - PhidgetsVoltage
  - PhidgetsCurrent (20 amps, DC)
- 設定檔內容：

```
port: /dev/ttyUSB0
baud: 57600
timeout: 0.1

rate: 50
sensorstate_rate: 10

use_base_controller: False
base_controller_rate: 10

# === Robot drivetrain parameters
#wheel_diameter: 0.146
#wheel_track: 0.2969
#encoder_resolution: 8384 # from Pololu for 131:1 motors
#gear_reduction: 1.0
#motors_reversed: True

# === PID parameters
#Kp: 20
#Kd: 12
#Ki: 0
#Ko: 50
#accel_limit: 1.0

# === Sensor definitions.  Examples only - edit for your robot.
```

```
#    Sensor type can be one of the follow (case sensitive!):
#        * Ping
#        * GP2D12
#        * Analog
#        * Digital
#        * PololuMotorCurrent
#        * PhidgetsVoltage
#        * PhidgetsCurrent (20 Amp, DC)

sensors: {
  #motor_current_left:   {pin: 0, type: PololuMotorCurrent, rate: 5},
  #motor_current_right:  {pin: 1, type: PololuMotorCurrent, rate: 5},
  #ir_front_center:     {pin: 2, type: GP2D12, rate: 10},
  #sonar_front_center:   {pin: 5, type: Ping, rate: 10},
  arduino_led:         {pin: 13, type: Digital, rate: 5, direction:
output}
}
```

示例開機檔案：

```
<launch>
  <node name="arduino" pkg="ros_arduino_python" type="arduino_node.py"
output="screen">
    <rosparam file="$(find
ros_arduino_python)/config/my_arduino_params.yaml" command="load" />
  </node>
</launch>
```

使用說明：

- 驅動程式需要 Python 2.6.5 或更高版本和 PySerial 2.3 或更高版本。
- 已經在 Ubuntu Linux 10.04（Maveric）和 11.10（Oneric）上測試。