```
#include <Servo.h>
Servo sv; //Servoオブジェクト“sv”を作成する
```
RCサーボを使用するためのライブラリをインクルードして
RCサーボを使えるようにする.

```
#include <PID_v1.h>
double Setpoint, Input, Output;
PID myPID(&Input, &Output, &Setpoint, 5.0, 0.3, 0.3, REVERSE);
```
..... Arduino PIDライブラリのインクルード
//ぶつからないPID制御用変数宣言
//ぶつからないPIDパラメタセッティング
P項 I項 D項 … 前車にうまく追従するように適宜調整する.

```
int thr = 900; //センサ閾値(右)
int thc = 900; //センサ閾値(中央)
int thl = 900; //センサ閾値(左)
```
ライン・センサ しきい値の設定. 線の検出ができるように調整する.

```
int strcntrconst = 90; //ステアリングセンター指令値
int fullstrconst = 25; //Δフル転舵角度（度）
int halfstrconst = 6;  //Δ中間転舵角度（度）

float fullspeed = 160; //最大速度指令値（０～２５５）

int strcommand = strcntrconst;
float speed = 0;       //走行用モータ速度指令値（０～２５５）
```
…… ハンドル操作：中立の指令値. 車両が直進するように調整
……「強旋回」の操作量
……「弱旋回」の操作量
コースに合わせて調整する.
…… 車両の走行速度の設定. コースに合わせて調整する.

```
void setup() {

  pinMode(2, OUTPUT); //右センサモニタLED
  pinMode(3, OUTPUT); //中央センサモニタLED
  pinMode(4, OUTPUT); //左センサモニタLED


  pinMode(5, OUTPUT); // TB67H450FNG IN2用出力ピン
  pinMode(6, OUTPUT); // TB67H450FNG IN1用出力ピン


  //initialize the variables we're linked to
  Setpoint = 250;      //前車との距離指定[mm]

  //set sampling time
  myPID.SetSampleTime(40);

  //set PID output limit
  myPID.SetOutputLimits(-1 * fullspeed, fullspeed);

  //turn the PID on
  myPID.SetMode(AUTOMATIC);

  Serial.begin(9600);

  sv.attach(9); //svの出力を~~D5~~9番ピンに割り当てる

}
```
…… 車両距離 目標値. 単位 mmで設定.

走行用モータ PID制御用の初期設定.

…… RCサーボの制御出力ピンを9ピンに設定する.

```
void loop() {

  //センサ読み取り

  int right  = analogRead(3); //A3:ラインセンサ右読取
  int center = analogRead(4); //A4:ラインセンサ中読取
  int left   = analogRead(5); //A5:ラインセンサ左読取


  //センサステータス表示ＬＥＤ駆動＆ステータスビット設定

  int br = 0; //status LSB:right sensor
  int bc = 0; //status bit:center sensor
  int bl = 0; //status MSB:left sensor

  if (right > thr) {
    br = 1;
    digitalWrite(2, HIGH);
  }

  else {
    br = 0;
    digitalWrite(2, LOW);
  }

  if (center > thc) {
    bc = 2;
    digitalWrite(3, HIGH);
  }

  else {
    bc = 0;
    digitalWrite(3, LOW);
  }
```
ライン・センサ・ユニットの出力電圧を読む

・ライン・センサ・ユニット出力電圧から
車両の状態を表1で示すとおりに
番号をつける.

・ライン・センサ・ユニット読み取り状況を
表示するLEDを点灯/消灯する.

```
if (left > thl) {
  bl = 4;
  digitalWrite(4, HIGH);
}

else {
  bl = 0;
  digitalWrite(4, LOW);
}

int result = br + bc + bl;
```

↳ 変数「result」に車両状態を示す番号が代入される．（車両の状態認知）

```
// 転舵量・速度の決定

switch (result) {
  case 0: // 線なしの場合 000

    break;

  case 1: // 強右旋回 001
    strcommand = strcntrconst - fullstrconst;
    speed = fullspeed;
    break;

  case 2: //  (センサ異常)  010
    speed = 0;
    break;

  case 3: // 弱右旋回 011
    strcommand = strcntrconst - halfstrconst;
    speed = fullspeed;
    break;

  case 4: // 強左旋回 100
    strcommand = strcntrconst + fullstrconst;
    speed = fullspeed;
    break;

  case 5: //  (センサ異常) 101
    speed = 0;
    break;

  case 6: // 弱左旋回 110
    strcommand = strcntrconst + halfstrconst;
    speed = fullspeed;
    break;

  case 7: // 直進  111
    strcommand = strcntrconst;
    speed = fullspeed;
    break;

  default:
    strcommand = strcntrconst;
}
```

・車両状態を示す変数「result」に応じて図10に示したとおりにハンドルの操作量を決定する．（ハンドル操作量の「判断」）

・ハンドル操作量は変数「strcommand」に代入．

```
//ステアリングサーボドライブ

sv.write( strcommand ); //0～180(deg)で指定
```

…… ステアリングサーボにハンドル操作量を書き込む（ハンドル「操作」の実行）

```
//  ぶつからない処理付モータドライブ

double distance = analogRead(0);

distance = 5 * distance / 1023;          // distanceをセンサ出力実電圧へ変換
Input = 266.88 * pow(distance, -1.265); // センサ出力実電圧を距離に変換（データシート記載グラフから読み取り）
Input = constrain(Input, 100, 800);      // 距離の変換結果を100～800mmの範囲にリミッティング
```

…… 車両距離の「認知」

```
myPID.Compute();
```

…… 走行用モータ操作量の「判断」

```
if (Output > 0) {

  digitalWrite(5, LOW);    //IN2
  analogWrite (6, Output); //IN1

}

else {

  analogWrite (5, -1 * Output); //IN2
  digitalWrite(6, LOW);         //IN1
```

走行用モータの「操作」

```
  }

  //シリアルモニタ出力
  Serial.print(Output);
  Serial.print("  ");
  Serial.print(left); //ラインセンサ左　読取値
  Serial.print("  ");
  Serial.print(center);//ラインセンサ中　読取値
  Serial.print("  ");
  Serial.println(right);//ラインセンサ右　読取値
}
```

ライン・センサ・ユニット読み取り値を
Arduino IDEでモニタできるように
シリアル通信で送信する.