令和1年度　人と地域共創センター公開講座（秋・冬）

# AI/IoTセンサのしくみを知ろう（応用編）

前方

電源スイッチ

## 第５回　ロボットの制御１（モーターとセンサーの協調動作）

川上　博

2019/11/16

# 講座内容

・講師：辻　明典（徳島大学技術支援部）
　　桑折範彦（徳島大学名誉教授）
　　川上　博（徳島大学名誉教授）

・土曜日：10:00～11:30

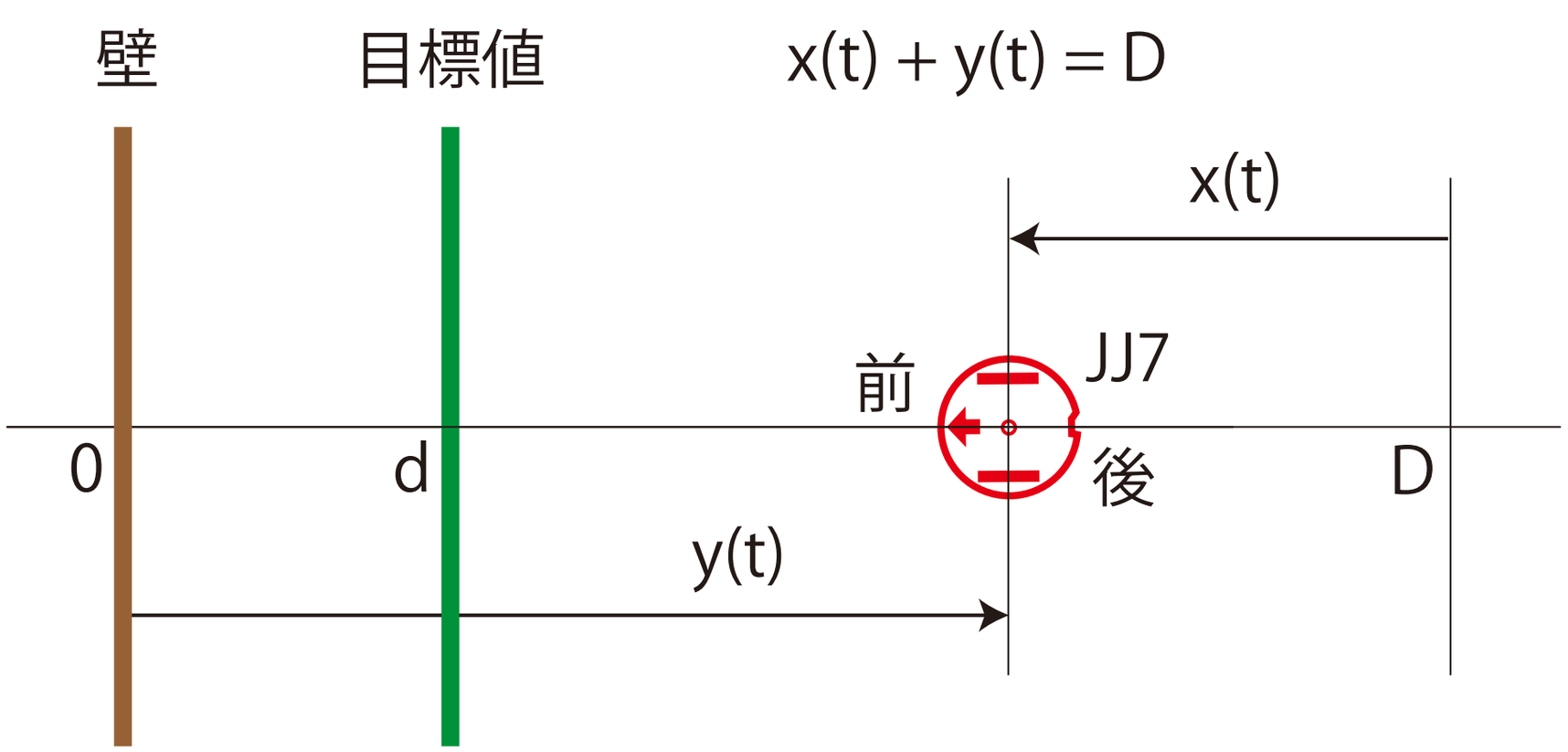
・日程：

❶ 10/5 概要，環境設定，配布部品の確認　　❷ 10/19 復習
❸ 10/26 ロボットのモーター１　　❹ 11/9 ロボットのモーター２
❺ 11/16 ロボットのセンサー１　　❻ 11/30 ロボットのセンサー２
❼ 12/7 ロボットの制御１　　❽ 12/14 ロボットの制御２
❾ 12/21 ロボットの制御３

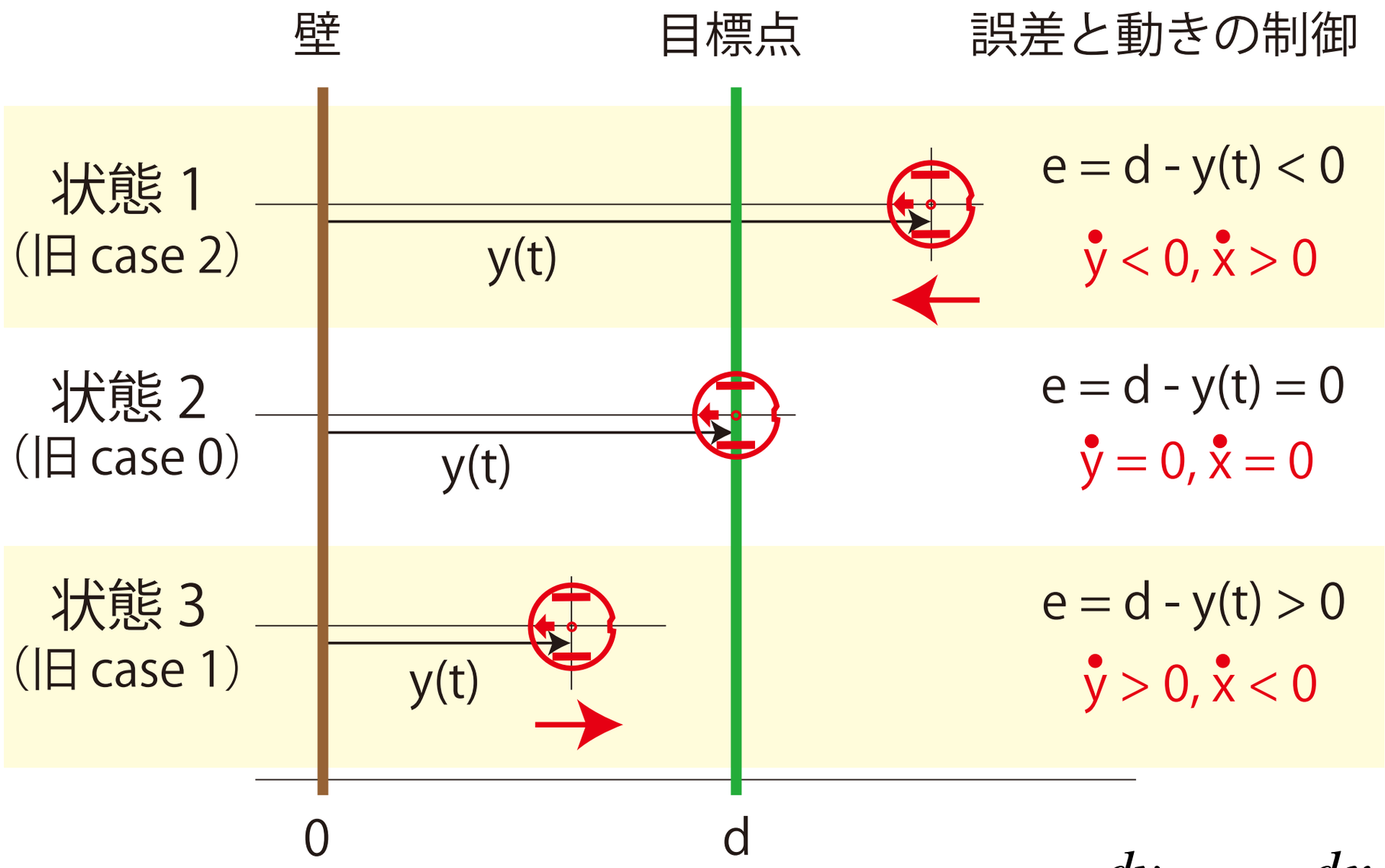
# 今日のテーマ：JJ7ロボットの定位置制御

---

壁に直交した方向に動くことができるようにJJ7が置いてある．JJ7には距離センサーがあり，JJ7ロボットの中心と壁との距離 y は常時測定できる．JJ7を壁から d[mm]の位置で止めておくにはどのように運転すればよいか？



# 運転法：誤差 e = d - y = 0 となるよう前後に動かす




$$x(t) + y(t) = D \qquad \frac{dy}{dt} = -\frac{dx}{dt}$$

# スケッチの主要な繰り返し部分：Ex0501～Ex0503

10[ms]毎に壁までの距離を求めyに代入
ただし，0 <= y <= 500[mm]とする

```
void loop() {
  int y = dist_measure(10, 0, 500);
  if (y != -1) {
    Serial.println(y);
    int s = robot_state(y); // ロボットの状態1, 2, 3
    robot_act(s); // ロボットの動作
  }
}

const float KP = 10.0; // 比例定数
float d = 100; // 目標距離 100[mm]
int u; // 制御量
int sp=300;
```

# robot_state 関数とrobot_act 関数

```
int robot_state(float y) {
  int s;

  u = KP * (d - y);
  if (u > sp) u = sp;
  if (u < -sp) u = -sp;
  Serial.println(r);
  if (u == 0) {
    s = 0; // 停止
  }
  else if (u > 0) {
    s = 1; // 後退
  }
  else if (u < 0) {
    s = 2; // 前進
  }

  return s;
}
```

```
void robot_act(int s) {
  switch (s) {
    case 0: // 停止
      motor(0, 0, HIGH, HIGH);
      break;
    case 1: // 後退
      motor((int)u, (int)u, LOW, LOW);
      break;
    case 2: // 前進
      motor((int) -u, (int) -u, HIGH, HIGH);
      break;
    default: // 停止
      motor(0, 0, HIGH, HIGH);
      break;
  }
}
```

```
velocityJJ7(0.0, -u);
```

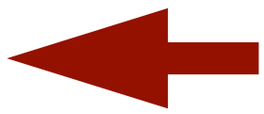
# JJ7 ロボットの定位置制御：Ex0504

```
void loop() {
  int y = dist_measure(10, 0, 500);
  if (y != -1) {// 0[mm]から500[mm]までの距離を10[ms]毎測定
    Serial.println(y);
    JJ7Controller(y); // ロボットを負帰還比例制御
    delay(10);          // 信号により前後に動かせる
  }
}
```

# JJ7Controller 関数

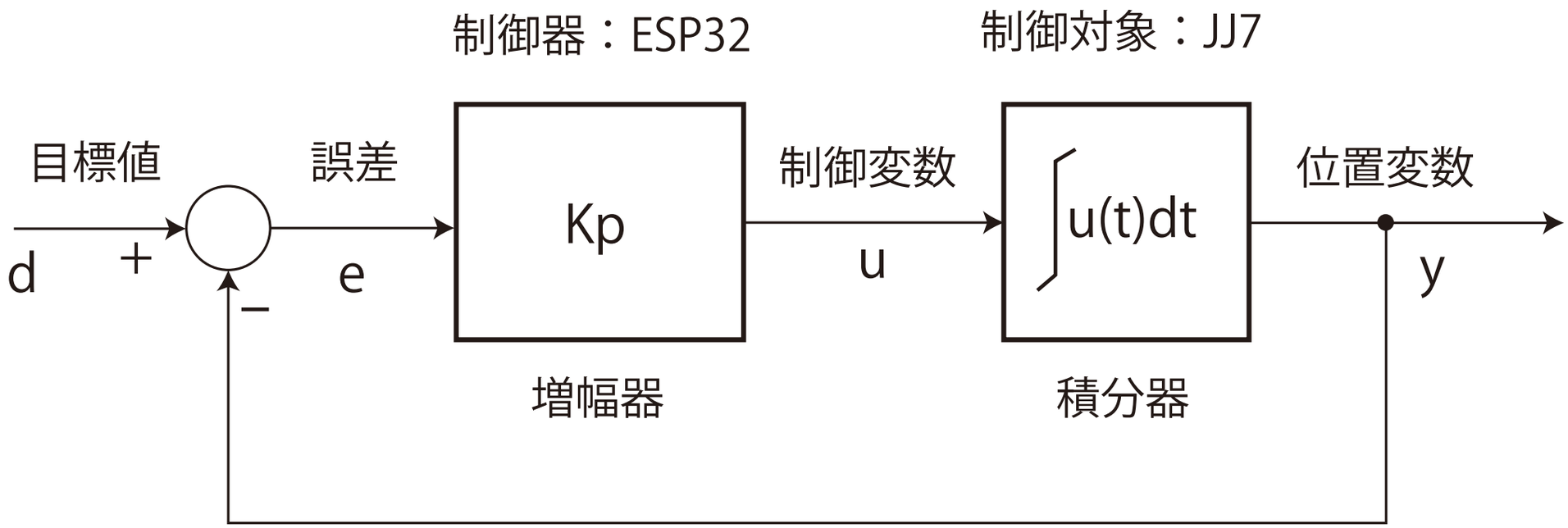
```
// 制御関係
const int sp=300;
float KP = 10.0, d = 100;

void JJ7Controller(float y) {
  float u; // 制御量
  u = KP * (d - y); // 比例制御変数（制御信号）
  if (u > sp) u = sp;  // 設定速度を超過
  if (u < -sp) u = -sp;// 設定速度を超過
  Serial.println(u);
  velocityJJ7(0.0, -u);
}
```



速度を与える式はこれ一つになってしまった！

# JJ7定位置制御のブロック線図



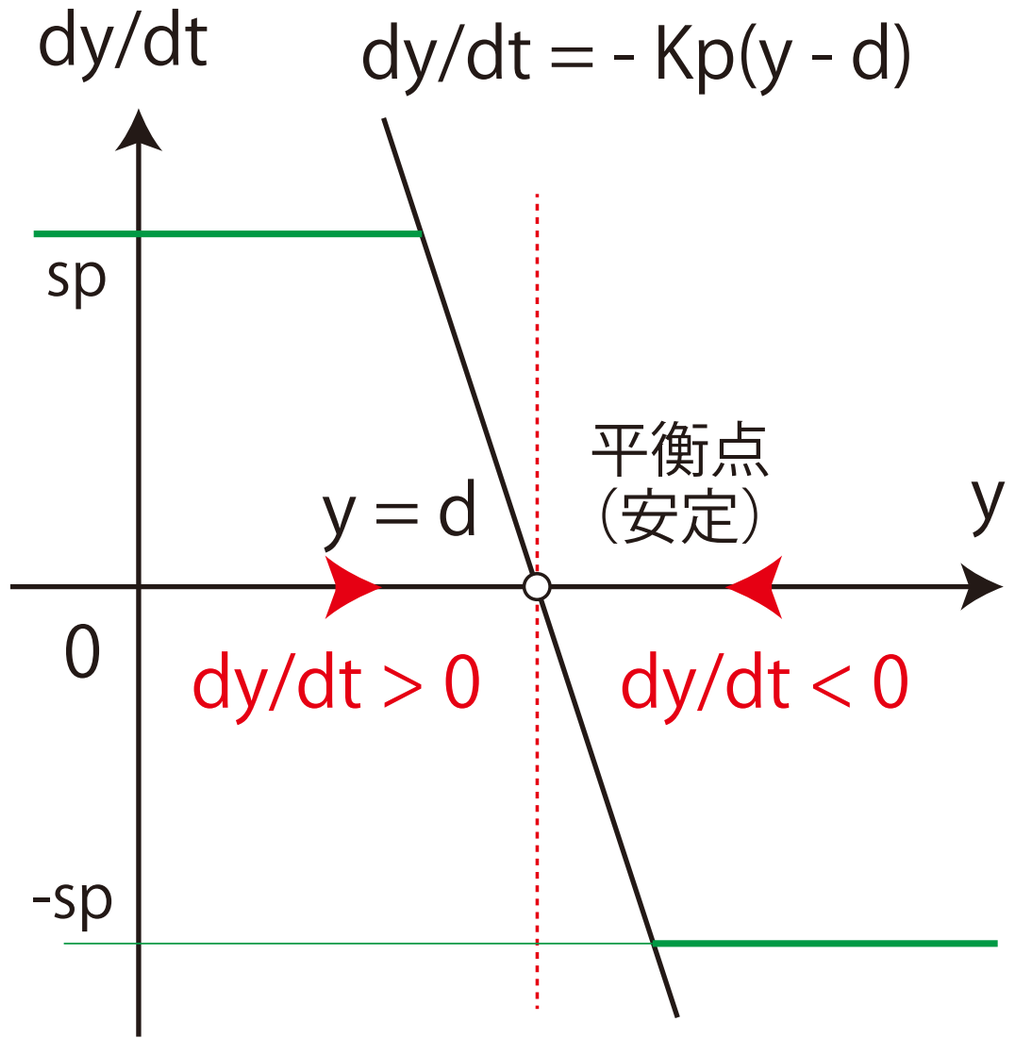

負帰還路( negative feed back loop)

$$e = d - y$$

$$u = K_p e = K_p(d - y)$$

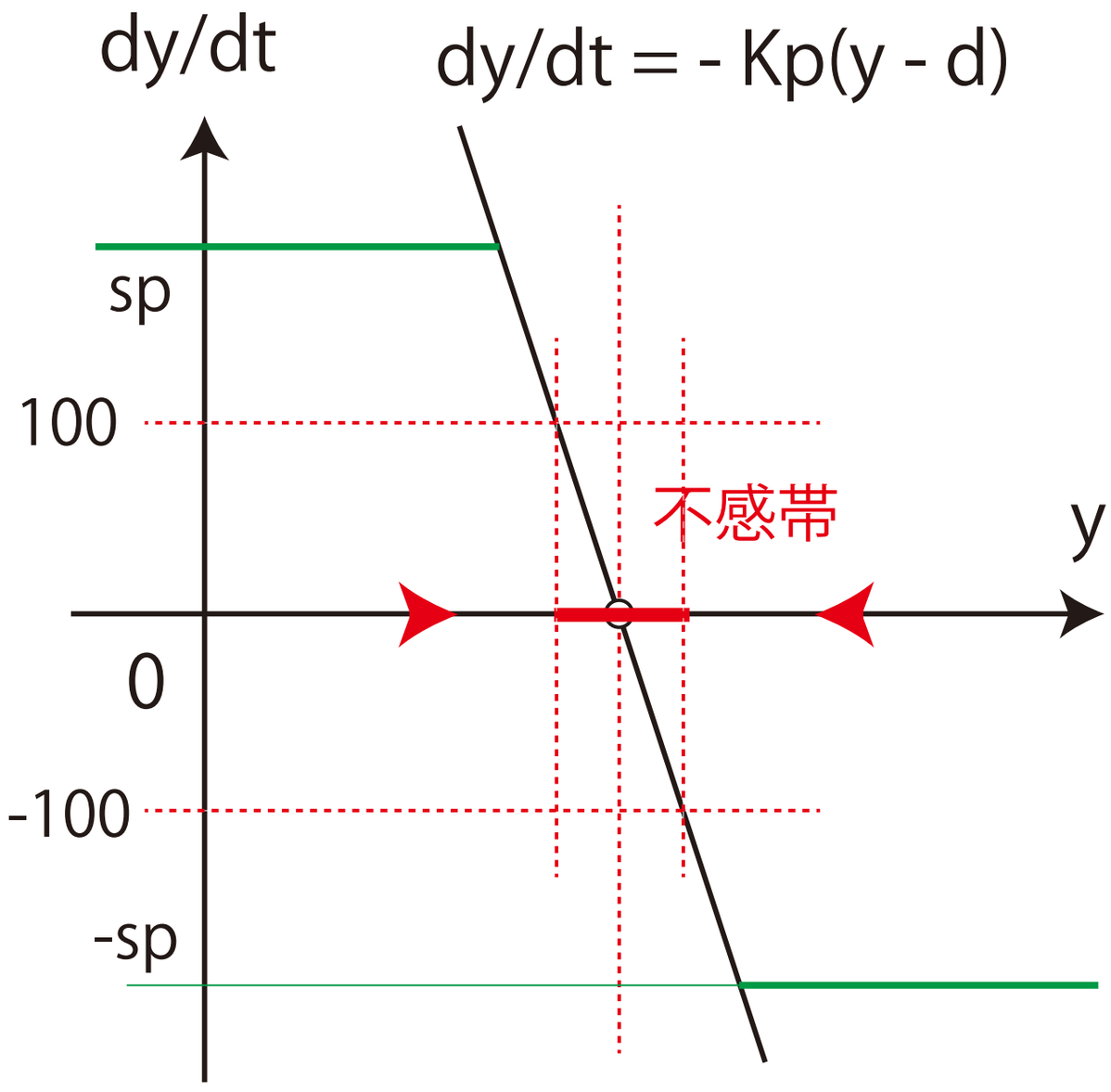
$$\frac{dy}{dt} = u = K_p(d - y)$$

# この制御系の振る舞いは？




任意の初期値 y0 から出発する解は平衡点に落ち着く

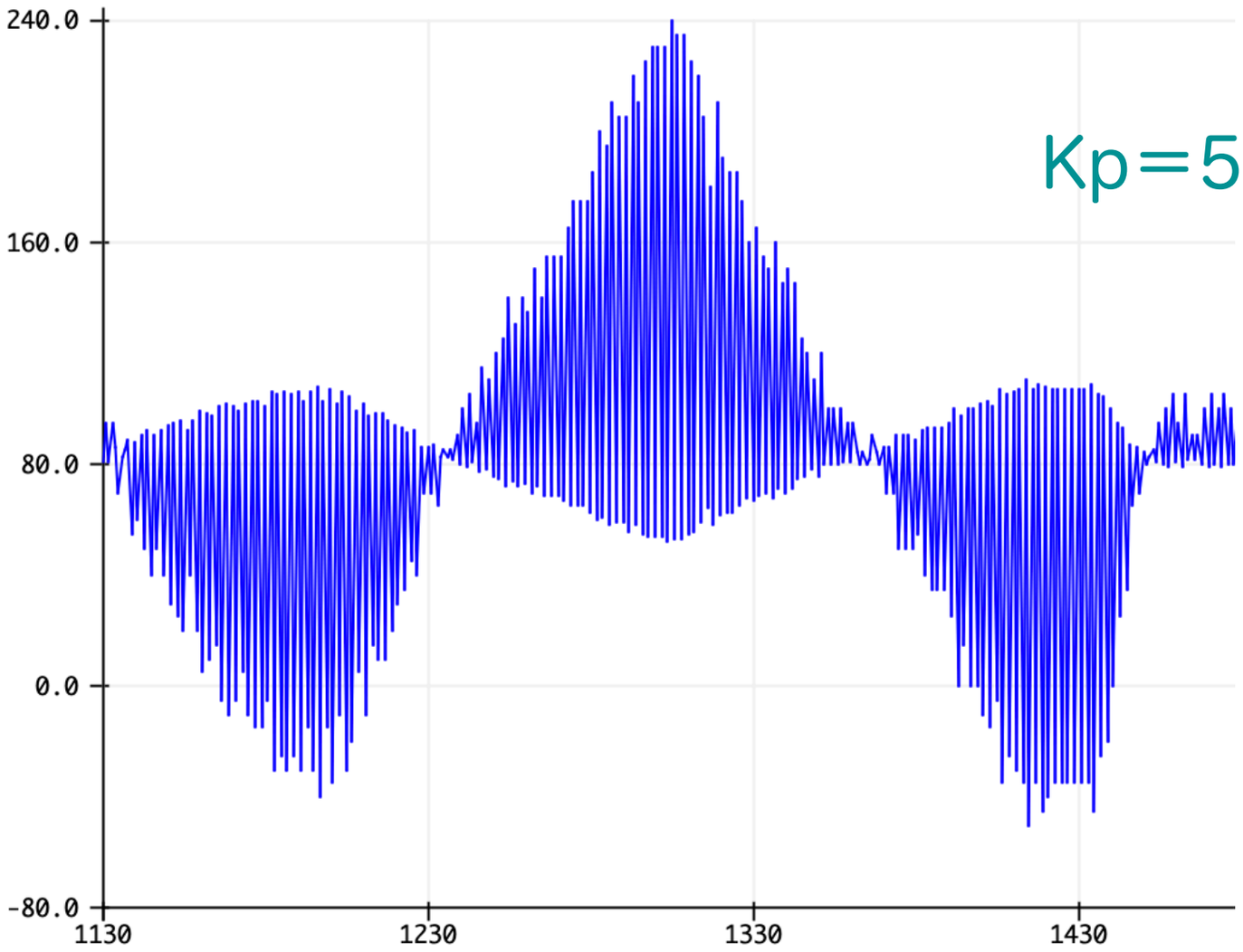
# 非線形効果に注意しよう




べったり平衡点が存在する．Kpを大きくすれば狭くなるが

# Kp の決め方・調整：限界感度法

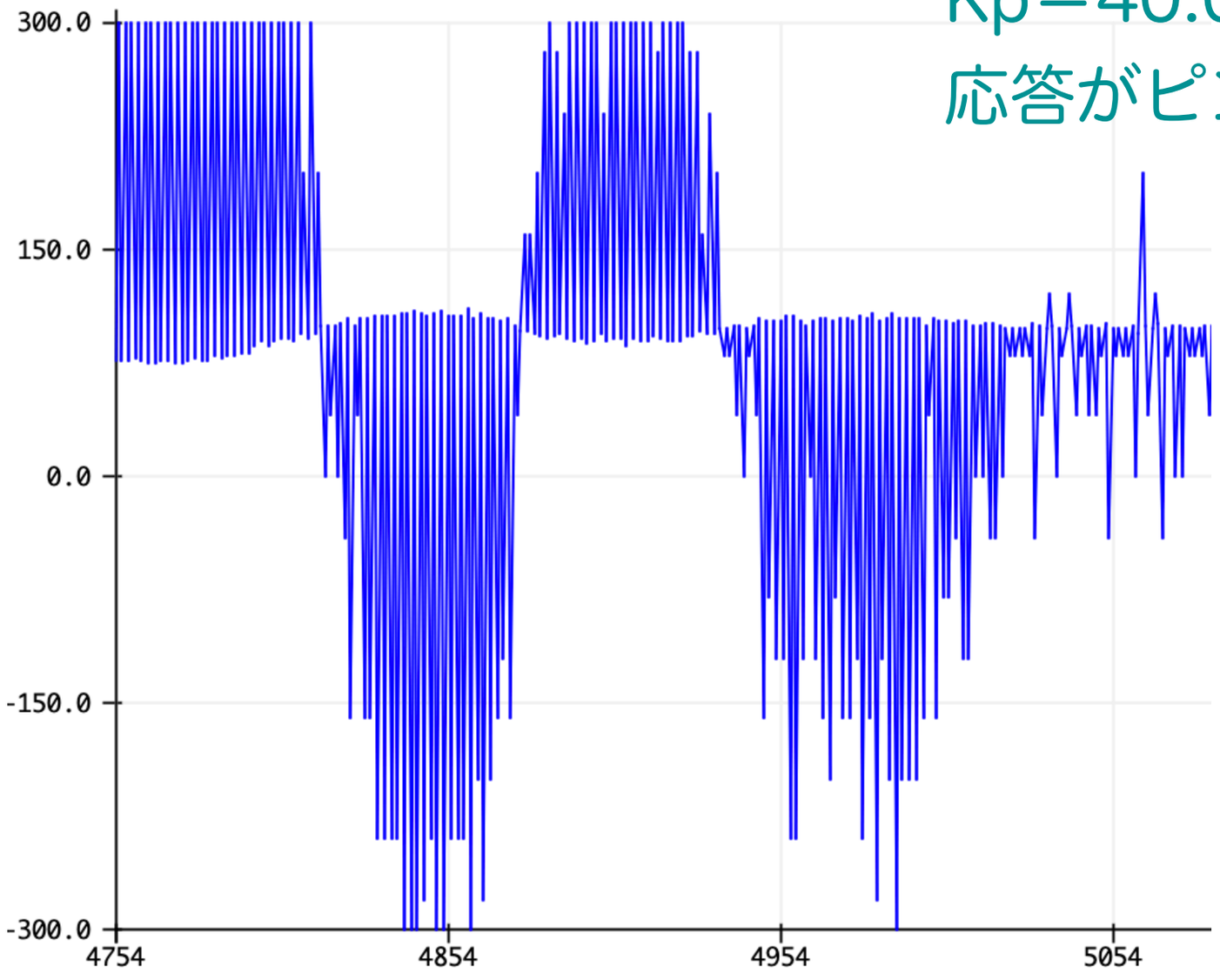
Kpを 5.0, 10.0, 20.0, 30.0, 40.0 と変化させて応答をみる.
振動的となる手前のKpを選ぶ.




Kp＝5.0程度の場合の応答

# Kp の決め方・調整：限界感度法

Kp＝40.0程度の場合の応答
応答がピンピンしている？



# Ex0505：安定平衡点が２つある制御系

```
const int sp=200;
float KP = 0.002, d1 = 50.0, d2=150.0, d3=250.0;

void JJ7Controller(float y) {
  float u; // 制御量
  u = KP * (d1 - y)*(d2-y)*(d3-y);
  if (u > sp) u = sp;
  if (u < -sp) u = -sp;
  Serial.println(u);
  velocityJJ7(0.0, -u);
}
```

制御量を決める関数 u が３次方程式となり
３つの平衡点があることに注意しよう

# 今日のまとめ

- Ex0501, Ex0502, Ex0503：これらのスケッチは，系が取るであろう状態を定義し，各状態が起こった時にどのような処理をするかをプログラムにした.
  有限状態機械を使った定式化：ディジタル的手法

- Ex0504：Kp を大きくすると u = sp or -sp のようなスイッチで制御しているような応答となる．JJ7は目標値を中心に激しく振動するので注意が必要である.

- Ex0504, Ex0505：負帰還制御系として扱い，スケッチをシンプルにまとめることができた.
  微分方程式を用いた定式化：アナログ的手法