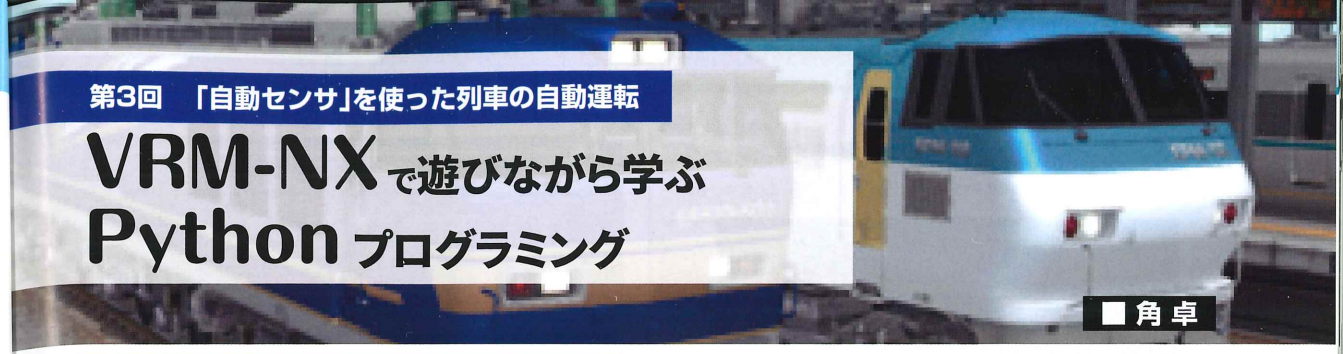


第3回　「自動センサ」を使った列車の自動運転

VRM-NXで遊びながら学ぶPythonプログラミング

■角卓

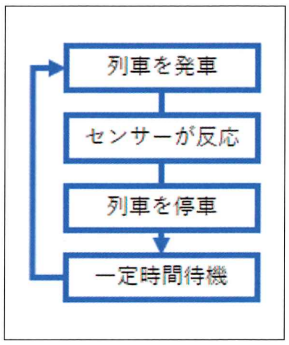
本記事ではPCソフト「鉄道模型シミュレーターNX」(VRM-NX)を使って、「Python」でソフトの「3D鉄道モデル」を動かすためのテクニックを紹介します。
プログラミングと鉄道が好きな人にオススメです。

「VRM-NX」で列車の自動運転

今回は「鉄道模型シミュレーターNX」(以下：VRM-NX)で、列車の「発車」と「停車」を「自動センサ」パーツと「Python」を使って自動で行なう方法を紹介します。

*

線路に重ねて配置した「自動センサ」を列車が通ると減速して停車し、一定時間後に発車するというプログラムを繰り返し実行します。



自動運転のプログラム

レイアウトの準備

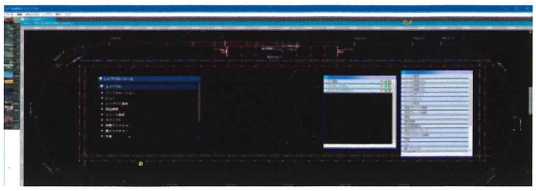
まずは列車を動かすレイアウトを準備します。

*

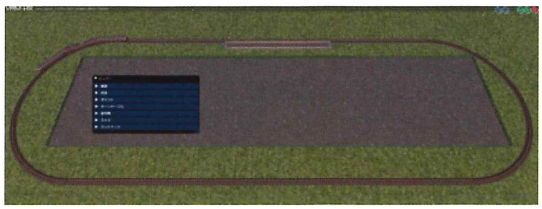
今回は直線と曲線を組み合わせた「環状線」(オーバル・レイアウト)を作ります。

上側に停車位置として駅ホームを配置し、駅ホームの左側、時計回りの進行方向手前に「自動センサ」を配置します。

列車は自動運転させるための1編成を時計回りになるように配置します。



オーバル・レイアウト



ビュワー表示

起動時に列車を発車させる

列車をビュワー起動時から走行させるようにするには「AutoSpeedCTRL」関数を使います。

「AutoSpeedCTRL」関数を使うと、指定した距離を走りながら速度を変化させます。
「第一引数」にはmm(ミリメートル)換算の「走行距離」、「第二引数」には「編成」に設定した「最高速度に対する速度比」を、0～1の浮動小数で指定します。

「AutoSpeedCTRL関数」の使用例

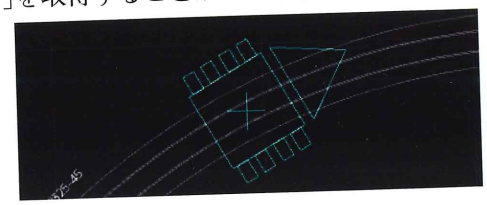
```
距離600mmで最高速度の75%に変更
VRMTrain.AutoSpeedCTRL(600.0, 0.75)
```

センサを反応させる

線路に重ねて配置した「自動センサ」パーツ(以下：センサ)は列車を検知すると「catchイベント」を発行します。

検知した列車はイベントの「obj引数」に対し

て「GetTrain」関数を使って「編成オブジェクト」を取得することができます。



「線路」に重ねた「センサ」

「自動センサ」での「編成オブジェクト」取得例

```
elif ev == 'catch':
    # 編成オブジェクトを取得
    tr = obj.GetTrain()
    # センサ検知ログ
    tn = tr.GetNAME()
    sn = obj.GetNAME()
    vrmapi.LOG(sn + "が" + tn + "検知")
```

列車を「停車」させる

列車の「停車」は「発車」と同様の「AutoSpeedCTRL」をセンサの「catchイベント」と組み合わせて使います。

停車位置を駅ホームに合わせたいときはセンサの位置を調節するか、「AutoSpeedCTRL」関数の走行距離を変更します。

列車を「待機・再出発」させる

列車を「停車」させてから一定時間経過後に発車させるにはセンサの「catchイベント」で「SetEventAfter」関数を使います。

列車の「afterイベント」に出発用の「AutoSpeedCTRL」関数を設定することで、「SetEventAfter」関数の引数で指定した時間の経過後に再出発します。

時間指定は「列車が停車してからの待機時間」ではなく、「センサが反応してからの経過時間」になります。

そのため、停車するまでの「減速時間」を含めた時間を設定します。

「Python」を書き込む

以上の内容を実現するためのスクリプトを列車とセンサにそれぞれ記載します。

列車用スクリプト（抜粋）

```
if ev == 'init':
    # 距離600mmで最高速度の75%に変更
    obj.AutoSpeedCTRL(600.0, 0.75)
elif ev == 'broadcast':
    dummy = 1
elif ev == 'timer':
    dummy = 1
elif ev == 'time':
    dummy = 1
elif ev == 'after':
    # 距離600mmで最高速度の75%に変更
    obj.AutoSpeedCTRL(600.0, 0.75)
```

センサ用スクリプト（抜粋）

```
elif ev == 'catch':
    # 編成オブジェクトを取得
    tr = obj.GetTrain()
    # 距離1100mmで速度を0
    tr.AutoSpeedCTRL(1100, 0.0)
    # 20秒後にafterイベント実行
    tr.SetEventAfter(20.0)
```

記述できたら、レイアウトを保存して、「運転ボタン」を押してみましょう。

ビュワー起動後に列車が走り出し、レイアウトを回ります。

センサを踏むと減速し、ホームへ停車します。

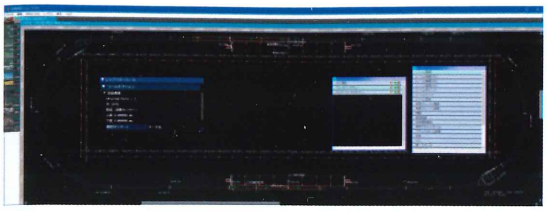
一定時間停車後にまた出発して、以降は同じ動作を繰り返します。

「列車」や「駅」を増やしてみる

この自動運転方法は「センサ」と「列車」が連携していますが、それぞれの仕組みは単体で結しています。

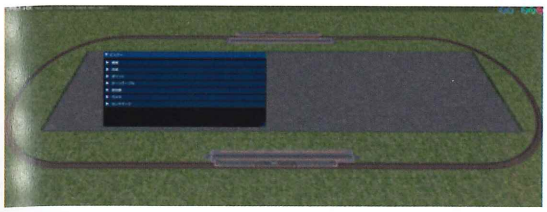
そのため、複製して増やすことができます。

「駅ホーム」「列車」「センサ」をレイアウト点対称となるように複製してみましょう。



ビュワーを起動すると、2つの列車が同じように走ります。

2編成の速度が同じで充分な距離が空いていれば、永久に走り続けます。



このように1つの「オーバル・レイアウト」内に同一行動の列車や停車位置を増やすことは簡単にできます。

1つの「オーバル・レイアウト」に複数の列車を配置して駅の停車を行なう場合は、駅停車中に後続の列車がぶつからないような間隔を空けて配置してください。

列車の密度を上げるには、実際の「鉄道運行計画」同様に、列車の速度を上げたり停車時間を短くしたりして調節しましょう。

おまけ：車両の電源をONにする

「VRM-NX」では列車の「ヘッドライト」や「ルームライト」などのON/OFF制御が可能です。

車両の各種電源ステータスは「編成オブジェクト」(VRMTrain)の中にある「車両オブジェクト」(VRMCar)に対して車両ごとに変更命令を記述する必要があります。

しかし、「レイアウトオブジェクト」の「GetTrainList」関数を利用することで、レイアウト内の全編成・全車両をまとめて属性変更することができます。

*

以下の内容を「レイアウト・オブジェクト」に記載すると、「ダミー編成以外の列車」の電灯やパンタグラフを「ON」にします。

全列車の電源をONにする

```
#LAYOUT
import vrmapi

def vrmevent(obj,ev,param):
    if ev == 'init':
        dummy = 1
        setAllCarPower()
    elif ev == 'broadcast':
        dummy = 1
    elif ev == 'timer':
        dummy = 1
    elif ev == 'time':
        dummy = 1
    elif ev == 'after':
```

⋮

「電源ON」(左)と「電源OFF」(右)

*

今回はセンサで「発車」と「停車」を繰り返す基本的な自動運転システムを作りました。

次回は「列車の識別」と「ポイント」を利用した「追い越しのある緩急ダイヤ」の自動運転について紹介する予定です。

※「鉄道模型シミュレーターNX」の仕様について最新情報を確認したい場合は、「VRM-NX SCRIPT MANUAL」(https://vrmcloud.net/nx/script/)で確認してください。

WEB版 ソースコード全文、画像