嵌入式智慧影像分析與實境界面
Fall 2021

Instructor：Yen-Lin Chen(陳彥霖), Ph.D.
Professor
Dept. Computer Science and Information Engineering
National Taipei University of Technology

下載NVIDIA-AI-IOT/jetbot

- $ cd ~
- $ mkdir Nvidia
- $ cd Nvidia
- $ git clone https://github.com/NVIDIA-AI-IOT/jetbot.git

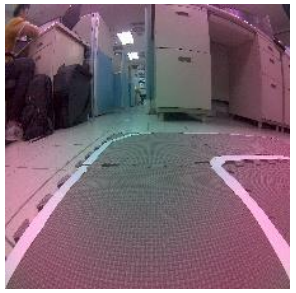
車道跟隨

- 使用程式檔案路徑: Nvidia/jetbot/notebook/road_following/
- 按照順序使用的檔案名稱:
  1. data_collection_gamepad.ipynb
  2. train_model.ipynb
  3. live_demo_build_trt.ipynb
  4. live_demo_trt.ipynb

實作流程

data_collection.ipynb

Input:224*224



收集車道影像，並進行標記影像。

train_model.ipynb

pytorch

Training (Resnet18)

使用pytorch訓練模型。

best_steering_model_xy.pth

live_demo_build_trt.ipynb

TensorRT

Optimization model

使用tensorRT來最佳化pytorch所訓練的模型。

best_steering_model_xy_trt.pth

live_demo_trt.ipynb

Live demo

透過相機輸入即時影像進行即時的車道辨識與Jetbot控制。

收集資料

data_collection_gamepad.ipynb

引用library

- 引用“資料收集”目的的函示庫。
- 我們將主要使用OpenCV來視覺化並保存含有標籤的影像。
- uuid, datetime之類的函示庫用於影像命名。

```
# IPython Libraries for display and widgets
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display

# Camera and Motor Interface for JetBot
from jetbot import Robot, Camera, bgr8_to_jpeg

# Python basic pakcages for image annotation
from uuid import uuid1
import os
import json
import glob
import datetime
import numpy as np
import cv2
import time
```

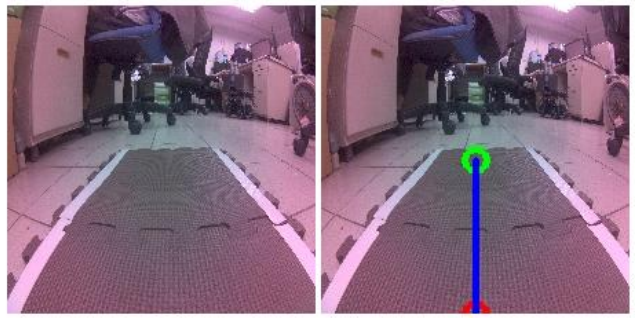
Python package，用來動態計算預設值與監視callback的功能。

提供一些功能，例如:滑桿、顯示影片等等。

用於顯示的API

定義相機與產生標記的物件

```
[2]: camera = Camera()

widget_width = camera.width
widget_height = camera.height

image_widget = widgets.Image(format='jpeg', width=widget_width, height=widget_height)
target_widget = widgets.Image(format='jpeg', width=widget_width, height=widget_height)

x_slider = widgets.FloatSlider(min=-1.0, max=1.0, step=0.001, description='x')
y_slider = widgets.FloatSlider(min=-1.0, max=1.0, step=0.001, description='y')

def display_xy(camera_image):
    image = np.copy(camera_image)
    x = x_slider.value
    y = y_slider.value
    x = int(x * widget_width / 2 + widget_width / 2)
    y = int(y * widget_height / 2 + widget_height / 2)
    image = cv2.circle(image, (x, y), 8, (0, 255, 0), 3)
    image = cv2.circle(image, (widget_width // 2, widget_height), 8, (0, 0,255), 3)
    image = cv2.line(image, (x,y), (widget_width // 2, widget_height), (255,0,0), 3)
    jpeg_image = bgr8_to_jpeg(image)
    return jpeg_image

time.sleep(1)
traitlets.dlink((camera, 'value'), (image_widget, 'value'), transform=bgr8_to_jpeg)
traitlets.dlink((camera, 'value'), (target_widget, 'value'), transform=display_xy)

display(widgets.HBox([image_widget, target_widget]), x_slider, y_slider)
```



x 0.00

y 0.00

- 使用JetBot的Camera Class來啟用CSI MIPI相機。
- 使用camera width, height(224x224)像素作為神經網路的輸入。

畫出Jetbot的位置與移動方向的位置

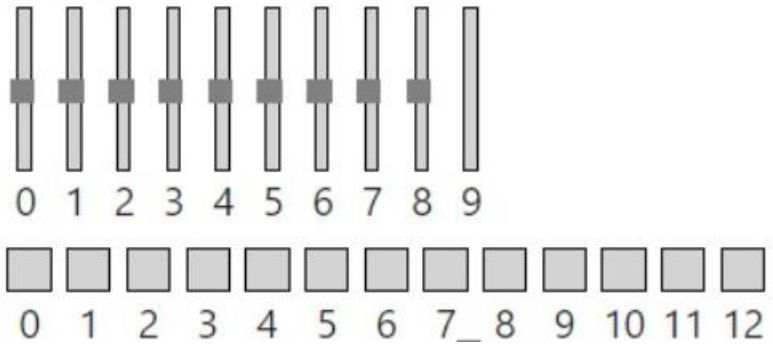
假設x_slider為-1時，-1 * 224 / 2 + 112 = 0
假設y_slider為-1時，-1 * 224 / 2 + 112 = 0
此時對應影像座標為(0, 0)

產生原始影像
使用display_xy控制x, y的位置，給予Jetbot移動方向

連接遙控器

- 此步驟類似於“teleoperation”，使用遊戲手把來標記影像。

```
controller = widgets.Controller(index=0)

display(controller)
```



設定控制x,y座標對應axis

- 設定搖桿axis代號，用來控制後續收集資料時x,y座標調整桿(axis代號請參考P.8)

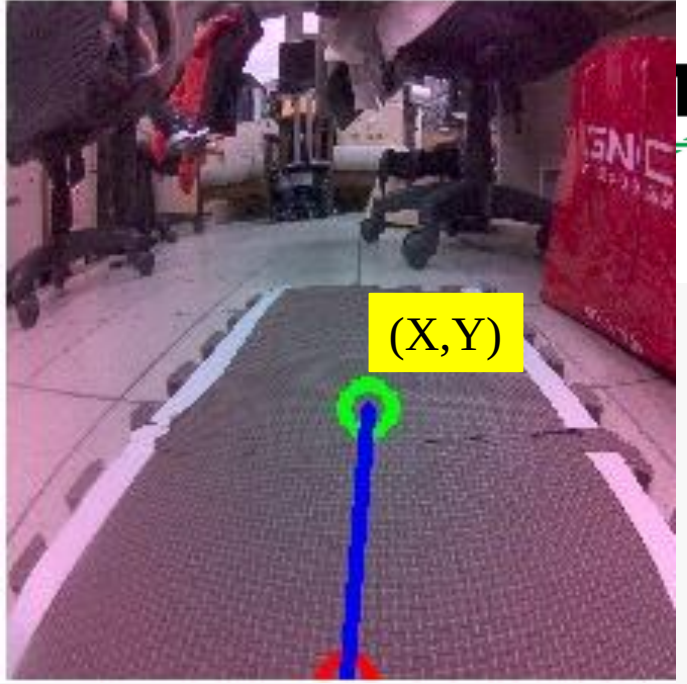
Connect Gamepad Controller to Label Images

Now, even though we've connected our gamepad, we haven't yet attached the controller to label images! We'll connect that to the left and right vertical axes using the dlink function. The dlink function, unlike the link function, allows us to attach a transform between the source and target.

[6]:
```
widgets.jsdlink((controller.axes[0], 'value'), (x_slider, 'value'))
widgets.jsdlink((controller.axes[2], 'value'), (y_slider, 'value'))
```

```
DirectionalLink(source=(Axis(value=-0.003921568393707275), 'value'), target=(FloatSlider(value=0.0, descriptio…
```

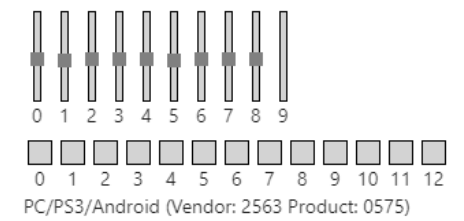
收集資料




```
DATASET_DIR = 'dataset_xy'

# we have this "try/except" statement because these next functions can throw an error if the d
try:
    os.makedirs(DATASET_DIR)
except FileExistsError:
    print('Directories not created becasue they already exist')

for b in controller.buttons:
    b.unobserve_all()

count_widget = widgets.IntText(description='count', value=len(glob.glob(os.path.join(DATASET_DIR, '*.jpg'))))

def xy_uuid(x, y):
    return 'xy_%03d_%03d_%s' % (x * 50 + 50, y * 50 + 50, uuid1())

def save_snapshot(change):
    if change['new']:
        uuid = xy_uuid(x_slider.value, y_slider.value)
        image_path = os.path.join(DATASET_DIR, uuid + '.jpg')
        with open(image_path, 'wb') as f:
            f.write(image_widget.value)
        count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))

controller.buttons[13].observe(save_snapshot, names='value')

display(widgets.VBox([
    target_widget,
    count_widget
]))
```

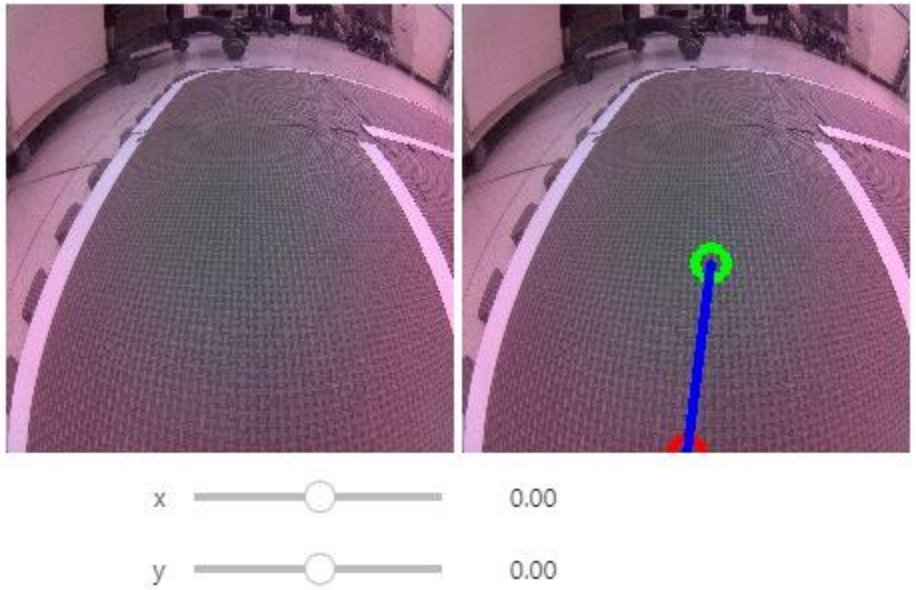
- 以下程式碼將顯示即時影像，以及我們已保存的影像數。
- 將Jetbot目前所視的影像，設定應該前往的目標(X, Y)，並將影像儲存為xy_<x value>_<y value>_<uuid>.jpg，作為此影像的標記。

※需自行更改想要使用的按鈕來記錄(controller按鍵代號請參考P.8)



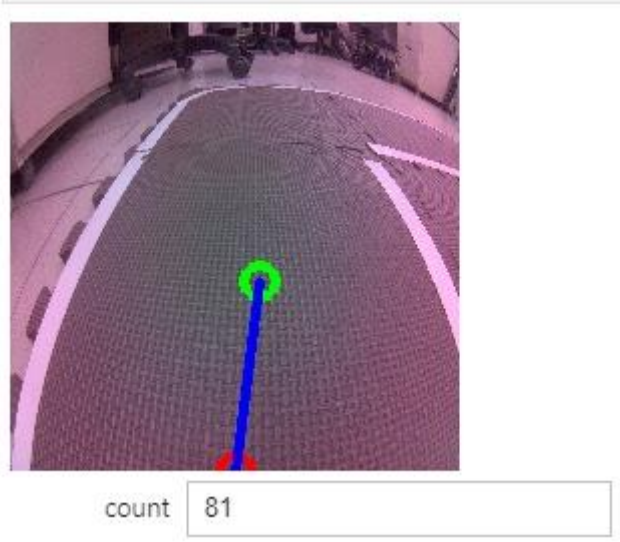
收集資料

Display live camera feed產生的視窗



- 透過滑桿或滑鼠游標拖曳來告訴Jetbot行駛的位置，按下設定的button，Jetbot會將照片存入資料夾中。

Collect data產生的視窗



- count為照片數量。

其他基本操作方法

停止相機:避免佔用到其他Jupyter的相機

```
camera.stop()
```

壓縮並保存收集的影像

```
def timestr():
    return str(datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S'))

!zip -r -q road_following_{DATASET_DIR}_{timestr()}.zip {DATASET_DIR}
```

訓練模型

train_model.ipynb

引用函式庫

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
import glob
import PIL.Image
import os
import numpy as np
```

引用Pytorch基本功能

Pytorch用於實現各種最佳化演算法的套件

卷積函數

用於自定義資料集

提供resnet18架構

將圖片轉為tensor

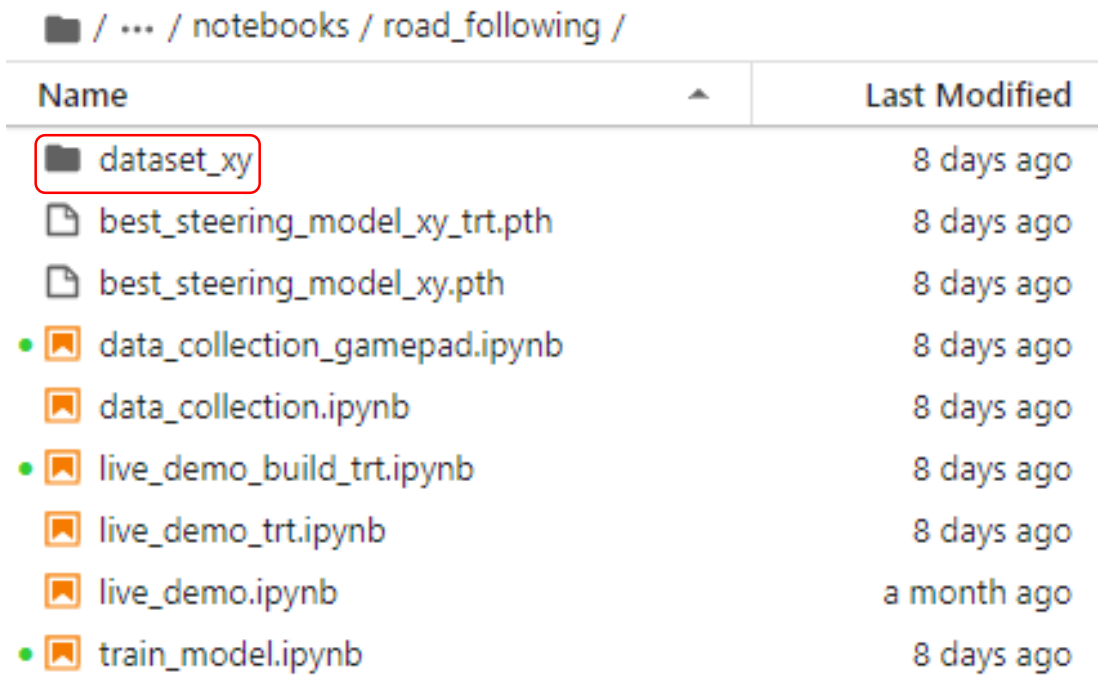
查詢檔案路徑

- 將使用PyTorch深度學習框架來訓練ResNet18神經網絡模型，以供道路跟隨之應用。

解壓縮檔案（有dataset_xy資料夾時忽略此步驟）

```
!unzip -q road_following.zip
```

- 前面的步驟已經產生了dataset_xy資料夾，且裡面包含了訓練資料集。
- 此步驟是將壓縮的檔案解壓縮後，來獲得dataset_xy資料夾，因為路徑中已經有資料夾了，所以這裡不需要解壓縮檔案。



torch.utils.data.Dataset

- PyTorch 透過torch.utils.data.Dataset類別，定義每一次訓練迭代的資料長相，例如：一張影像和一個標籤、一張影像和多個標籤...等，將所有資料打包起來，並送進torch.utils.data.DataLoader類別，定義如何取樣資料，以及使用多少資源來得到一個批次 (batch) 的資料，也可以讓使用者自定義資料集。
- 以下為官方提供的預設資料:
  - 影像辨識：MNIST(手寫數字)、Fashion-MNIST(衣著)、LSUN(物件、場景)、Imagenet(物件、場景)...。
  - 物件偵測：MS COCO、VOCDetection。
  - 影像分割：VOCSegmentation、Cityscapes
  - 標題產生：MS COCO、SBU、Flickr8k、Flickr30k

產生資料集格式

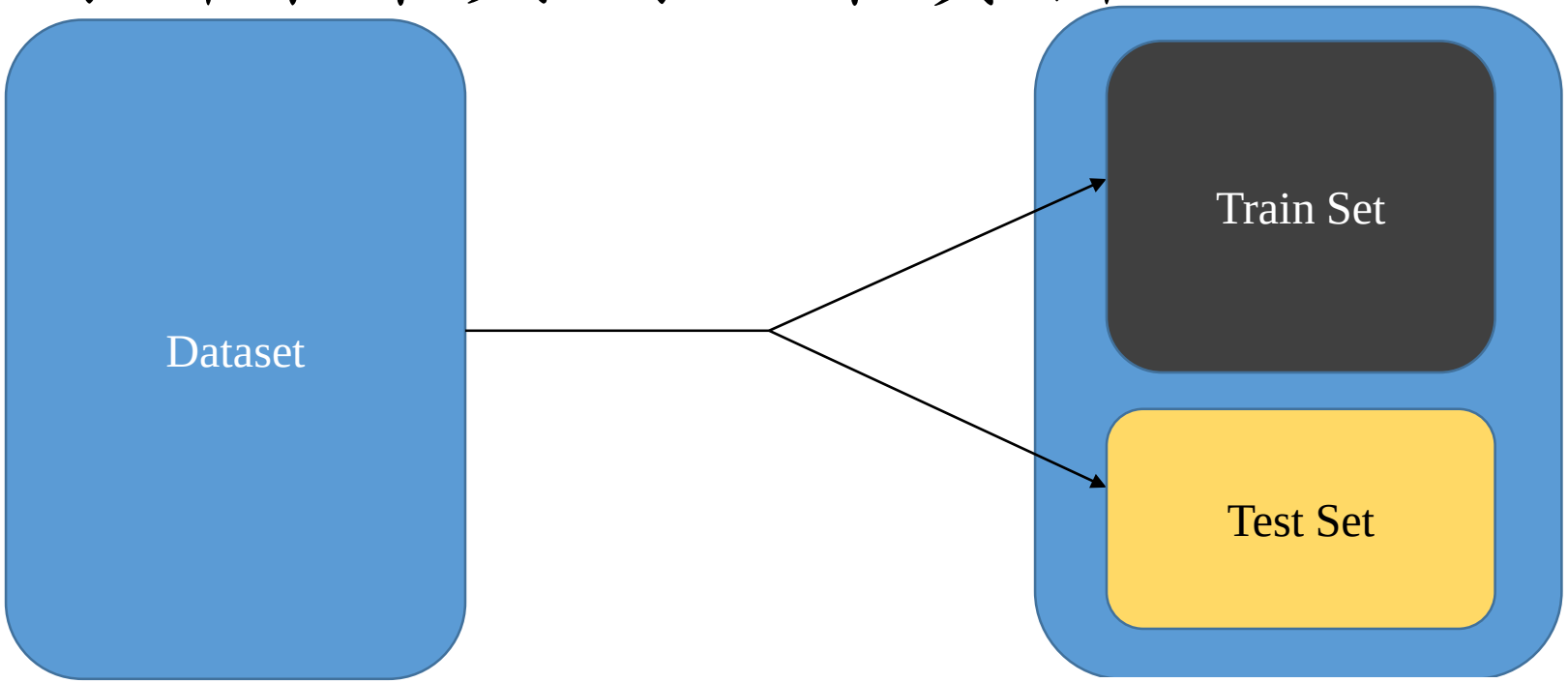
```python
def get_x(path):
    """Gets the x value from the image filename"""
    return (float(int(path[3:6])) - 50.0) / 50.0

def get_y(path):
    """Gets the y value from the image filename"""
    return (float(int(path[7:10])) - 50.0) / 50.0

class XYDataset(torch.utils.data.Dataset):

    def __init__(self, directory, random_hflips=False):
        self.directory = directory
        self.random_hflips = random_hflips
        self.image_paths = glob.glob(os.path.join(self.directory, '*.jpg'))
        self.color_jitter = transforms.ColorJitter(0.3, 0.3, 0.3, 0.3)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]

        image = PIL.Image.open(image_path)
        x = float(get_x(os.path.basename(image_path)))
        y = float(get_y(os.path.basename(image_path)))

        if float(np.random.rand(1)) > 0.5:
            image = transforms.functional.hflip(image)
            x = -x

        image = self.color_jitter(image)
        image = transforms.functional.resize(image, (224, 224))
        image = transforms.functional.to_tensor(image)
        image = image.numpy()[::-1].copy()
        image = torch.from_numpy(image)
        image = transforms.functional.normalize(image, [0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

        return image, torch.tensor([x, y]).float()

dataset = XYDataset('dataset_xy', random_hflips=False)
```

- Class XYDataset負責讀取影像並從影像文件名中取出x, y值。
- 這邊使用torch.utils.data.Dataset，並繼承這個class來自定義資料集

- __init__:資料的來源路徑、型態、索引等初始化資料集設定。
- __len__:設定資料集的大小。
- __getitem__:Override原本的讀取資料或前處理方法，並回傳資料給下一步的處理流程。

- random_hflip會將輸入影像進行隨機的水平翻轉，當設為true時可用來增加資料強度。

切割訓練集與測試集資料




```
test_percent = 0.1   取10%做為測試集
num_test = int(test_percent * len(dataset))
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - num_test, num_test])
```

- 讀取資料集後，將資料集隨機分為訓練集和測試集，會使用訓練集進行模型訓練後，用分離出來的測試及測量模型準確性的方法。
- 在專案範例中，拆分為訓練90%測試10%，各位同學可以自行調整比例，觀察在不同比例下，所訓練出來的模型準確度。

batch size、iteration、epoch觀念解釋

- **Batch Size**：每次訓練的樣本數量，batch Size太小會導致效率低下，無法收斂。Batch Size增大到一定程度後會導致裝置的記憶體撐不住，正確選擇batch size是為了取得記憶體使用效率和記憶體容量之間的平衡。
- **Iteration**：Iteration 是所有 batch 都完成一次訓練所需要的次數。
- **Epoch** ：一次的 epoch 代表樣本集內所有的資料都經過了一次訓練。在訓練過程中，每次epoch 都會對輸入的資料進行shuffle，並重新分成不同的batch。
- $1\ epoch = numbers\ of\ iteration = \frac{total\ example\ Nums}{batchSize}$
- 比如有一個 2000 個訓練樣本的資料集。將 2000 個樣本分成大小為 500 的 batch，那麼完成一次 epoch 需要 4 個 iteration。

設定訓練參數

```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)
```

Train_dataset : 訓練資料集。
Batch_size :訓練批次大小。
Shuffle :是否隨機順序讀取影像。
Num_works :設定執行緒，0代表只用單執行緒進行訓練。Nano最大可以設定到4，使用全部的核心進行訓練。

- 使用DataLoader來批次讀取資料，混淆資料並允許使用多個執行緒。
- Batch_size將影響到模型的最佳化程度和速度，範例將批次大小設定為8。批次處理大小是基於輸入影像大小與GPU實際可用的記憶體，設定值太大可能會造成訓練錯誤，太小訓練次數會變多，這會影響模型的準確性。

定義網路模型

- 專案範例是使用PyTorch TorchVision上的ResNet-18模型。

```
model = models.resnet18(pretrained=True)
```
使用resnet18 pretrain model

- 重新初始化model線性層, in_feature:512 out_feature:2
- 使用Nano上的GPU及Cuda進行模型訓練。

```
model.fc = torch.nn.Linear(512, 2)
device = torch.device('cuda')
model = model.to(device)
```

訓練資料集

```
NUM_EPOCHS = 70
BEST_MODEL_PATH = 'best_steering_model_xy.pth'
best_loss = 1e9

optimizer = optim.Adam(model.parameters())

for epoch in range(NUM_EPOCHS):

    model.train()
    train_loss = 0.0
    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.mse_loss(outputs, labels)
        train_loss += float(loss)
        loss.backward()
        optimizer.step()
    train_loss /= len(train_loader)

    model.eval()
    test_loss = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        loss = F.mse_loss(outputs, labels)
        test_loss += float(loss)
    test_loss /= len(test_loader)

    print('%f, %f' % (train_loss, test_loss))
    if test_loss < best_loss:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_loss = test_loss
```

訓練次數

讀取TensorRT最佳化後的模型

最佳loss值

使用Adam最佳化模型參數

訓練模式

讀取訓練資料集

評估模式

讀取測試資料集

- 設定訓練次數為70次，各組可以嘗試其他次數，確認其是否不同，並保存最佳模型。

複製images、labels到GPU記憶體中。

在Pytorch中避免前一次的訓練梯度結果影響到這次的訓練梯度。

使用這次iteration的資料進行訓練。

使用損失函數評估目前模型的好與壞。

統計每一次iteration的損失。

更新本次iteration訓練參數。

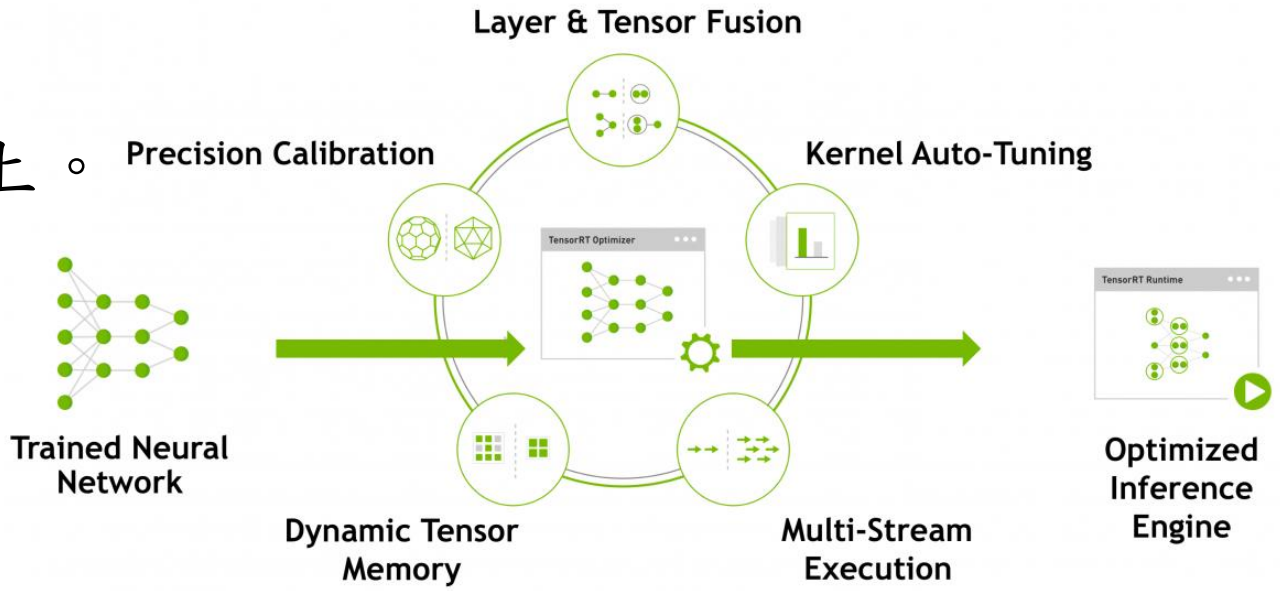
計算本次epoch的loss。

當模型損失小於預設損失時，儲存模型。

高效能深度學習開發套件
TensorRT

TensorRT 介紹

- TensorRT是一個高效能的深度學習推理平台，使用者可以將訓練好的類神經網路輸入TensorRT中，產出經最佳化後的推理引擎。
- 根據Nvidia的說法，使用TensorRT的應用程式，最快可以比CPU平台執行速度快40倍。
- 係由Nvidia開發的開源套件。
- 以C++撰寫，並建構於CUDA上。
- 可套用於PyTorch、MatLab、TensorFlow、Caffe 2等。

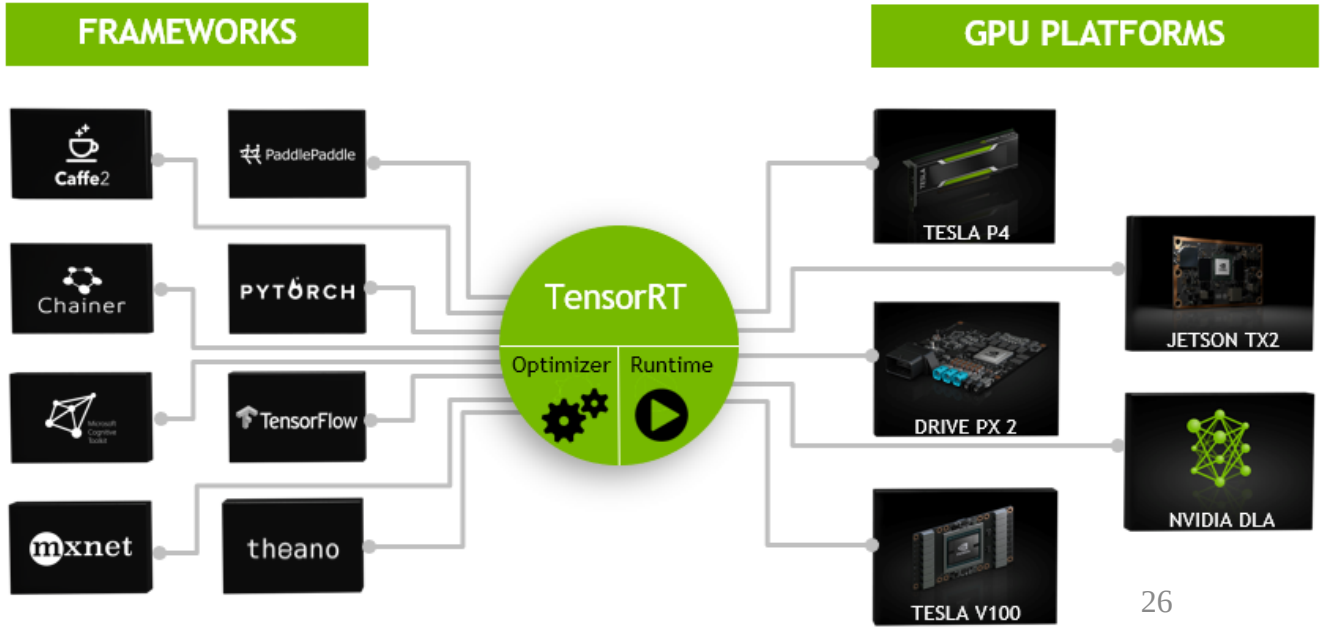


TensorRT 優勢

- 一旦神經網路訓練好了，TensorRT就能最佳化/部屬出一個執行順序，讓框架的負擔變輕。
- TensorRT能搭配神經網路層，將核心的選擇最佳化，同時也搭配正規化，並依照特定的精度(FP32, FP16, INT8) 最佳化矩陣數學運算來改進神經網路的延遲、輸入與輸出的流量和效率。
- 客製化一些應用程式來執行神經網路可能可以達到更高的效率，但會需要大量的人力與相對應於現代GPU的知識。甚至，在某個GPU上的最佳化不見得能完全轉移到其他GPU上，因為每一代GPU提供的功能不盡相同。但TensorRT透過API能解決這些問題。

TensorRT 的運作

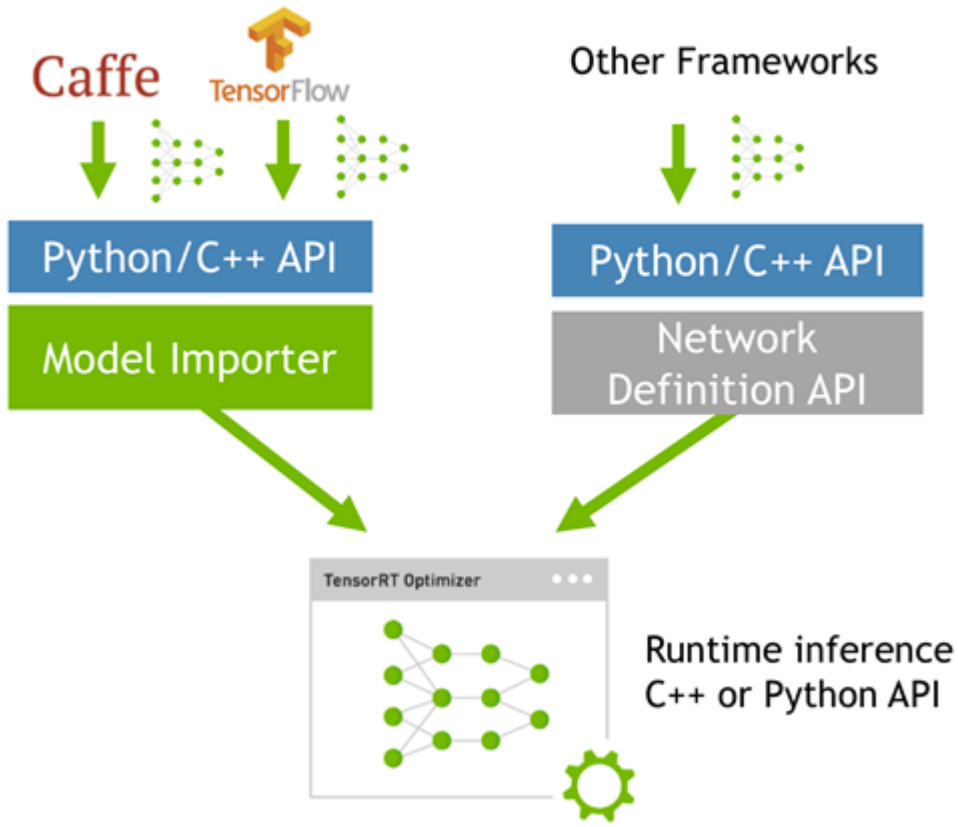
- 為了最佳化出神經網路模型，TensorRT會將網路架構與平台最佳化後產生一個Inference Engine，這個流程稱作建置階段。建置階段會花費大量時間，尤其是在嵌入式平台上。典型的神經網路只需要建置一次，接著存成Plan File以供之後使用。
- 產生的Plan File並不能跨平台使用，因為Plan File通常都是針對特定GPU所做，也必須在指定的GPU上執行。



TensorRT 的運作

在建置階段中，TensorRT對網路層所做的最佳化包含：

- 刪除輸出用不到的層數
- 融合卷積，包含bias與ReLU函數等
- 將相似的參數與相同來源的Tensor有效地彙整起來
- 融合串聯層，將層數的輸出直接指向該去的位址



TensorRT最佳化模型

live_demo_build_trt.ipynb

定義輸入模型格式

- 定義輸入模型格式，將模型設置為pretrained=False，目的是只使用網路架構，本專案目前使用resnet18，如有使用其他網路架構，則需要修改使用的網路。
- 根據訓練時的模型格式，並使用cuda、評估模式、float16，執行以下程式碼以讀取初始化PyTorch模型。

```
import torchvision
import torch

model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 2)
model = model.cuda().eval().half()
```

- 讀取要轉換成TensorRT的模型。

```
model.load_state_dict(torch.load('best_steering_model_xy.pth'))
```

```
device = torch.device('cuda')
```

TensorRT轉換最佳化模型

- 操作此步驟時，請同學先安裝TensorRT再來執行接下來的程式碼。(安裝後請重新開啟kernel再執行)

```
cd $HOME
git clone https://github.com/NVIDIA-AI-IOT/torch2trt
cd torch2trt
sudo python3 setup.py install
```

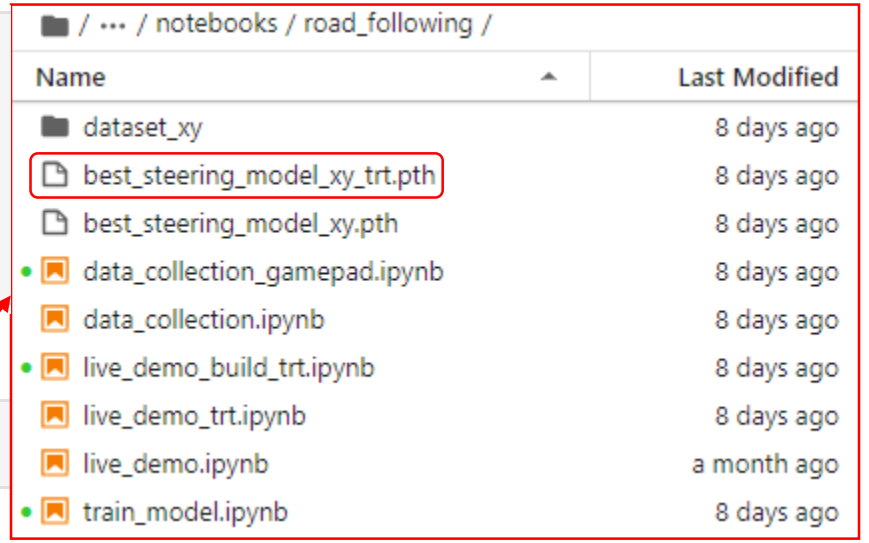
※請安裝TensorRT

- 使用torch2trt最佳化模型，以便使用TensorRT進行更快的判斷。

```
from torch2trt import torch2trt

data = torch.zeros((1, 3, 224, 224)).cuda().half()

model_trt = torch2trt(model, [data], fp16_mode=True)
```

- 儲存轉換好的模型。

```
torch.save(model_trt.state_dict(), 'best_steering_model_xy_trt.pth')
```



執行模型

live_demo_trt.ipynb

讀取模型

- 初始化pytorch，指定使用cuda加速。

```
import torch
device = torch.device('cuda')
```

- 將tensorRT模型透過TRTModule載入。

```
import torch
from torch2trt import TRTModule

model_trt = TRTModule()
model_trt.load_state_dict(torch.load('best_steering_model_xy_trt.pth'))
```

建立前處理功能

訓練模型所輸入的影像與相機擷取的色彩格式不同，因此需要進行一些前處理，需要以下步驟：

1.將影像從OpenCV的HWC格式轉換為CHW格式，C = Channel、H = Hegiht、W = width
2.將資料從CPU記憶體傳輸到GPU記憶體
3.使用與訓練期間相同的參數進行正規化

```
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np

mean = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()

def preprocess(image):
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device).half()
    image.sub_(mean[:, None, None]).div_(std[:, None, None])
    return image[None, ...]
```

對CHW分別減掉平均值，
再除以標準差來達到正規化。

建立即時影像視窗

- 透過獲得相機的資訊來顯示即時的影像視窗。

```
from IPython.display import display
import ipywidgets
import traitlets
from jetbot import Camera, bgr8_to_jpeg

camera = Camera()
```

定義相機

```
image_widget = ipywidgets.Image()

traitlets.dlink((camera, 'value'), (image_widget, 'value'), transform=bgr8_to_jpeg)

display(image_widget)
```

dlink()將原對象與目標對象的特徵連接起來

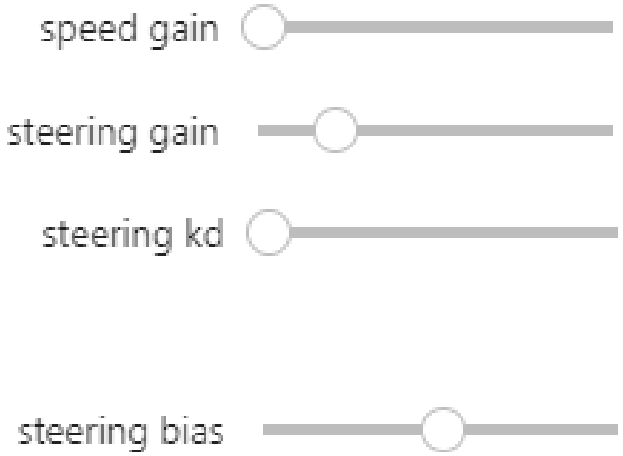
連接Jetbot

```
from jetbot import Robot

robot = Robot()
```

移動參數(自定義馬達增益參數)

- 透過調整參數來修正Jetbot的移動行為。

```
speed_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, description='speed gain')
steering_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, value=0.2, description='steering gain')
steering_dgain_slider = ipywidgets.FloatSlider(min=0.0, max=0.5, step=0.001, value=0.0, description='steering kd')
steering_bias_slider = ipywidgets.FloatSlider(min=-0.3, max=0.3, step=0.01, value=0.0, description='steering bias')

display(speed_gain_slider, steering_gain_slider, steering_dgain_slider, steering_bias_slider)
```



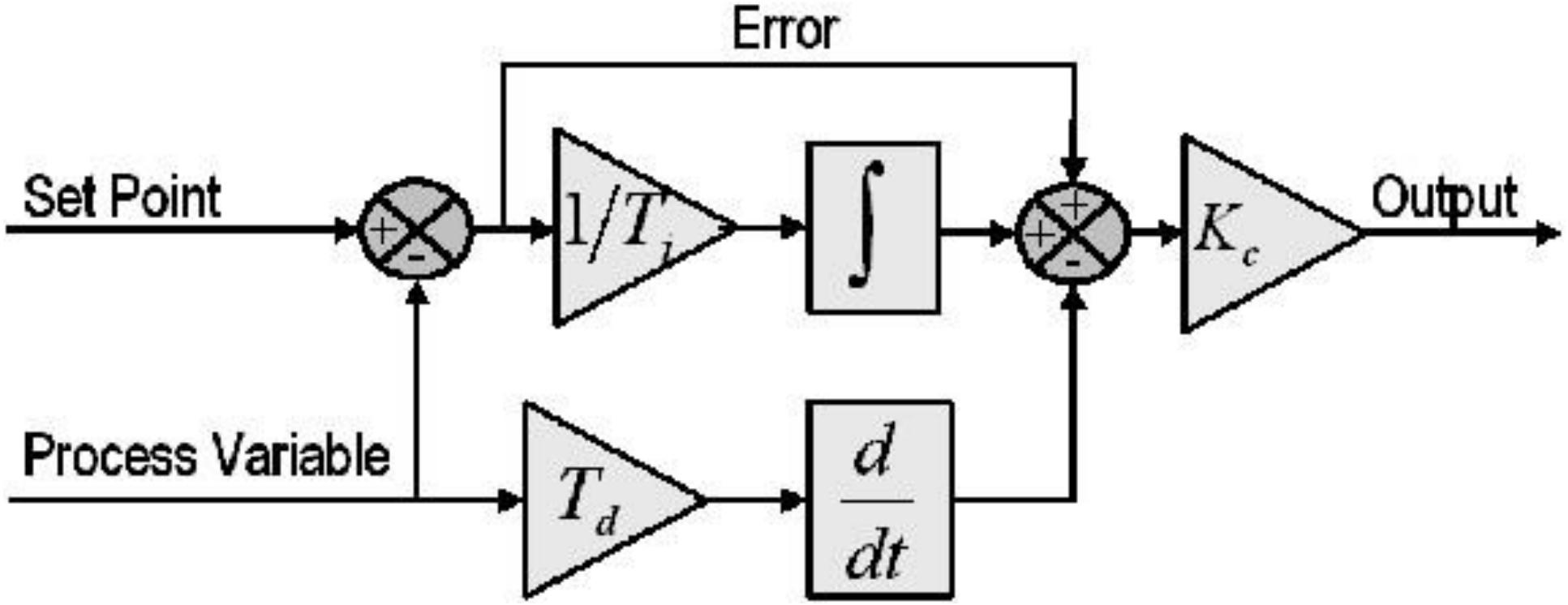
速度增益 (0~1)，設定Jetbot向前速度的基礎大小。

轉向增益(P) (0~1)，設定Jetbot對於旋轉角度的增益程度。

微分增益(D)，計算目前與前一次推論出的角度誤差增益，數值越大，PID越容易受到前一次偏移角度的影響。

轉向修正(-0.3~0.3)，由於Jetbot左右輪的馬達存在著公差，即使給予兩個馬達相同控制轉速功率(直行)，仍可能會因為公差發生偏轉，調整此值可修正Jetbot兩輪的輸出功率公差。

PID 控制原理



基本 PID 控制演算法的程式圖

PID 控制原理

- **比例響應(P)**
  PID 控制原理的第一個重要概念是比例響應。比例元件僅會因設定點與程序變數之間的差異而有所不同。 這項差異稱為「誤差項」。 「比例增益」(K_c) 決定了錯誤訊號對輸出響應的比例。例如，當誤差項程度達到 10 時，5 的比例增益會產生 50 的比例響應。一般而言，增加比例增益會同時增加控制系統響應速度。不過，當比例增益太大時，程序變數就會開始震盪。 當 K_c 進一步增加時，震盪幅度會變得更大，系統也會變得不穩定，甚至造成震盪失控。
- 在本專案用於設定Jetbot的轉向大小比例響應。

PID 控制原理

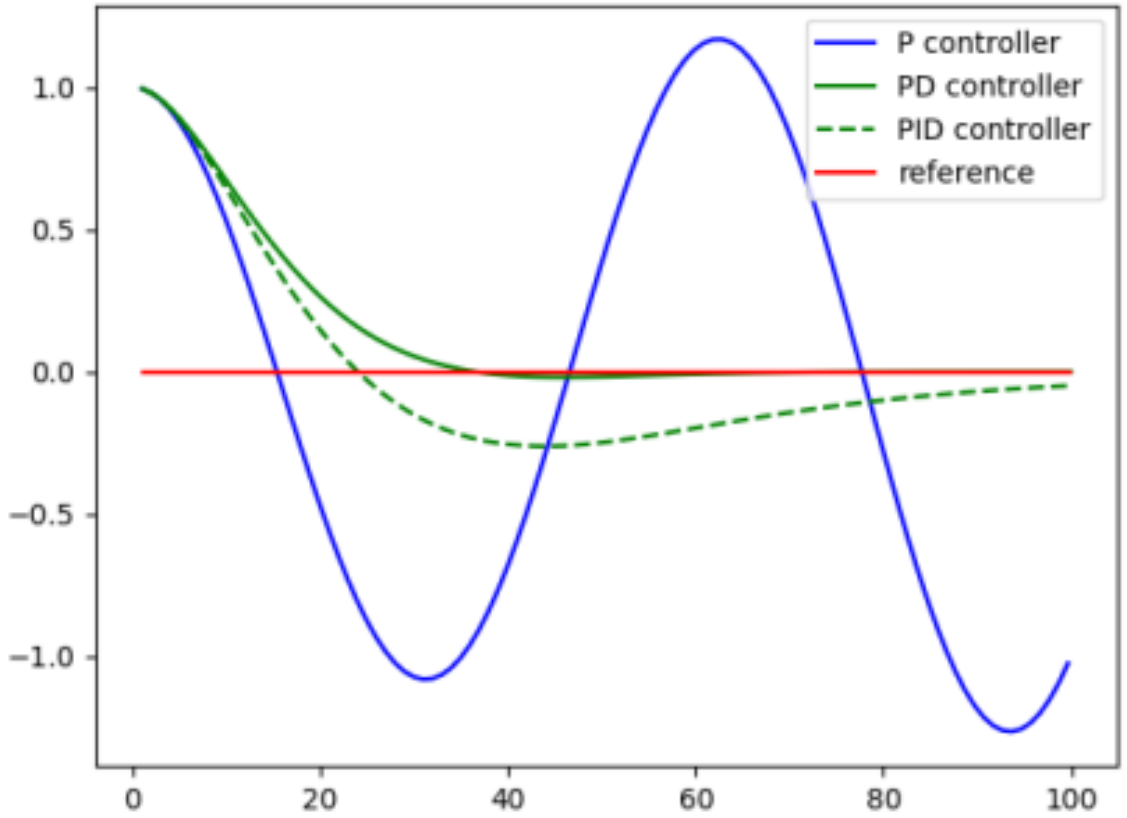
- **積分響應(I)**
  積分響應是了解 PID 控制教學中的第二個重要概念。積分元件會在一段時間後將誤差項加總。結果會變成就算是小小的誤差項，也會導致積分元件緩慢增加。 積分響應會隨著時間經過持續增加（但誤差為零時則例外），因此其影響在於讓穩態誤差趨近於零。「穩態誤差」指的是程序變數與設定點之間的最終差異。 當積分動作滿足控制器的過程中，不會導致控制器使誤差訊號趨於零時，就會產生稱為「積分終結」的現象。
- 在本專案只使用PD進行轉向控制。

PID 控制原理

- **微分響應(D)**
  PID 控制原理的第三個重要概念是微分響應。當程序變數快速增加時，微分元件會讓輸出減少。 微分響應與程序變數的變更速率成比例增減。 增加微分時間 (T_d) 參數會導致控制系統對誤差項中的變化反應更加強烈，而且會增加整體控制系統響應的速度。大多數實務上的控制系統都使用很小的微分時間 (T_d)，這是因為微分響應對於程序變數訊號中的雜訊非常敏感。 當感測器反饋訊號出現雜訊，或當控制迴圈速率太慢，微分響應會讓控制系統變得不穩定。
- 在本專案用於模型所推論的前一次偏移角度對於目前偏移角度的影響比例。

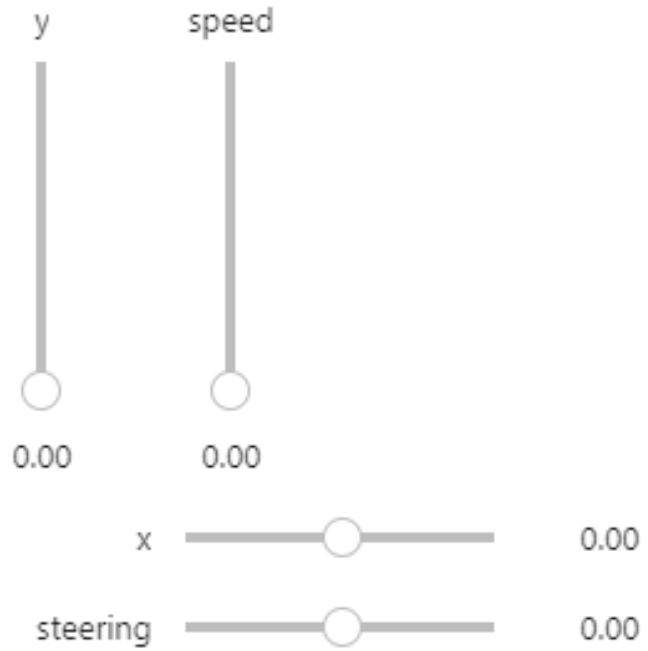
PID 控制原理

- PID可以僅使用P、PD、PID等三種型式，以下為使用三種不同方式的控制反饋圖。



Jetbot預測參數可視化界面

```
x_slider = ipywidgets.FloatSlider(min=-1.0, max=1.0, description='x')
y_slider = ipywidgets.FloatSlider(min=0, max=1.0, orientation='vertical', description='y')
steering_slider = ipywidgets.FloatSlider(min=-1.0, max=1.0, description='steering')
speed_slider = ipywidgets.FloatSlider(min=0, max=1.0, orientation='vertical', description='speed')

display(ipywidgets.HBox([y_slider, speed_slider]))
display(x_slider, steering_slider)
```



- 滑軌將顯示即時的轉向、速度及xy值。
- x, y : 為Jetbot要前往的座標位置。
- Steering : 當該值為負值時，Jetbot向左轉。
  當該值為正值時，Jetbot向右轉。
- Speed : Jetbot馬達給予的速度。

- 請記住，這些值並不是代表的實際角度，而是相對的比例值。
- 當需要轉向的角度為0時，Steering將為0，並且隨著需要轉向的角度而增加/減少。

Jetbot 控制原理

- Jetbot預設下會提供左右輪一個固定的向前速度(speed_slider)，使用PID同時調整可油門與轉向，油門與轉向角度可以用一個比例公式的概念串起來。當調整旋轉角度時，油門就會根據轉向角度來作左右輪油門的加減，如下圖，轉向大小(steering_slider)為0時，Jetbot左右輪油門為(speed_slider)，此時Jetbot理想上會向前行走；在其它狀況下，給予的轉向角度(steering_slider)愈大，左右輪的油門速差就愈大，相對的Jetbot旋轉速度就會越大，當

```
robot.left_motor.value = max(min(speed_slider.value + steering_slider.value, 1.0), 0.0)
robot.right_motor.value = max(min(speed_slider.value - steering_slider.value, 1.0), 0.0)
```

Jetbot PID轉向角度計算與公差修正

```
angle = np.arctan2(x, y)
pid = angle * steering_gain_slider.value + (angle - angle_last) * steering_dgain_slider.value
angle_last = angle

steering_slider.value = pid + steering_bias_slider.value
```

$$angle = argtan\frac{x}{y}$$

計算目前模型所推論出Jetbot與道路兩線的偏移角度。

$$pid = angle \times Gain_{steering} + (angle - angle\_last) \times Gain_{\Delta steering}$$

用目前偏移角度乘上轉向增益可得到要讓Jetbot轉向的大小(P)，如目前與前一次的轉向角度偏移差過大，並加上目前偏移角度與前一次偏移角度的差值乘上增益大小進行轉向的修正(D)。

$$steering = pid + Bias_{steering}$$

用兩個馬達的偏差值進行公差修正，得到最後的的轉向大小。

執行

- 透過execute function來使用最佳化模型進行、並設定自定義馬達增益參數。

```
angle = 0.0
angle_last = 0.0

def execute(change):
    global angle, angle_last
    image = change['new']
    xy = model_trt(preprocess(image)).detach().float().cpu().numpy().flatten()
    x = xy[0]
    y = (0.5 - xy[1]) / 2.0#取出model所推論出目前畫面中Jetbot要前往的x, y值位置

    x_slider.value = x
    y_slider.value = y
    #將x, y值帶入前面定義的滑軌參數x_slider, y_slider，提供可視化結果
    speed_slider.value = speed_gain_slider.value
    #將speed_gain_slider(速度增益)值帶入前面定義的滑軌參數speed_slider
    angle = np.arctan2(x, y)
    pid = angle * steering_gain_slider.value + (angle - angle_last) * steering_dgain_slider.value
    angle_last = angle

    steering_slider.value = pid + steering_bias_slider.value
    # steering_slider轉向參數=steering_bias_slider(修正參數)+角度
    robot.left_motor.value = max(min(speed_slider.value + steering_slider.value, 1.0), 0.0)
    robot.right_motor.value = max(min(speed_slider.value - steering_slider.value, 1.0), 0.0)
    #速度增益大小±轉向大小，給予左右輪相反的轉向大小，透過兩個輪子轉速速差，讓Jetbot發生轉向。
execute({'new': camera.value})
```

⬅ ※建議先把數值調小

注意!
執行此程式，Jetbot會開始移動，但還沒載入相機的資訊，所以請先抓好你的Jetbot，避免暴衝。

載入相機資訊

- 透過camera.observe( )來讓Jetbot擁有相機資訊。
- 接下來，Jetbot會跟著道路行駛!

```
camera.observe(execute, names='value')
```

停止並關閉相機

```
import time

camera.unobserve(execute, names='value')  停止輸入相機的資訊

time.sleep(0.1)  # add a small sleep to make sure frames have finished processing

robot.stop()   停止Jetbot
```

```
camera.stop()  釋放相機資源
```

- 如果想要停止Jetbot的動作可以執行此段程式碼。

Project demo

- https://www.youtube.com/watch?app=desktop&v=ivGtBA3DiSs&ab_channel=%E6%9D%8E%E6%8C%AF%E8%B1%AA

Project 3

AI人工智慧之道路辨識

專案實作-AI道路辨識

- 專案項目:車道線模型訓練與辨識
- 各小組告報中，應包含訓練的次數、訓練的圖像張數、Jetbot辨識之狀況、如何修正Jetbot，讓車道辨識更穩定且快速。
- 本專案應於Jetbot上實作。

小組報告格式規定

- 專案情境
  - 小組所討論出來的議題，並簡明扼要描述議題的情境。
- 定義問題
  - 將議題中的問題定義出來，並收斂問題方向。
- 方案構思
  - 簡單描述如何解決定義好的問題，並預計使用的技術。
- 解決方法
  - 說明實際上如何完成此議題的方案構思。
- 成果影片(僅小組報告)
  - 以影片方式呈現，並上傳至youtube後，將連結遷入至小組報告最後一頁。
- 分工
  - 說明小組成員分工內容與比例。

個人報告內容

- 個人報告內容須要有以下內容：
  - 你一開始所提出的議題是?你的議題是否有被選為小組議題候選?
  - 你在小組議題中，提出了那些問題與解決方案?是否有被小組接受?
    - 如個人所提出的方案沒被接受，是因為那些原因?
    - 為了這個議題，你去找了那些資料?你是如何分析找到的資料?
    - 其他小組成員所提出的提議有哪些?而你對於其他人的提議意見如何?
  - 在小組決定小組議題過程中，你對於小組最後提出的議題討論是否能接受?接受理由為何?不接受理由為何?
  - 你是否能接受最後的議題與方案?如接受請說明接受與否的理由?
  - 本次專案個人的心得
  - 本次你認為小組成員的貢獻比例及理由

專案繳交規則

- 小組報告繳交期限:110/11/22 23:59(以I學園上傳時間為基準)
- 個人報告繳交期限:110/11/22 23:59(以I學園上傳時間為基準)
- 補交規則
  - 超過正常繳交期限兩周內成績**打8折**
  - 超過正常繳交期限兩周後不接受補交