# DonkeyCar代码整理

本文档主要整理GitHub项目【1】中donkeycar V2.2.1版本的代码。采用自顶向下的分析方法，模拟程序的跳转流程依次分析。

## 测试脚本：

|  |
| --- |
| from donkeycar import Vehicle  from donkeycar.parts.camera import PiCamera  from donkeycar.parts.datastore import Tub  V = Vehicle()  #add a camera part  cam = PiCamera()  V.add(cam, outputs=['image'], threaded=True)  #add tub part to record images  tub = Tub(path='~/d2/gettings\_started',  inputs=['image'],  types=['image\_array'])  V.add(tub, inputs=['image'])  #start the drive loop at 10 Hz  V.start(rate\_hz=10) |

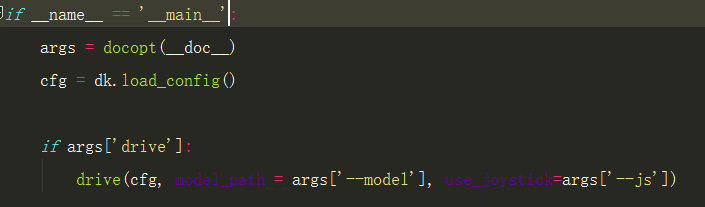


# 程序入口manage.py

安装好小车之后，进入系统，用命令donkey createcar –path ~/d2在本地文件夹下创建d2文件夹，这就是小车的根目录，建好之后会有一个manage.py文件，就是小车的启动入口。

通过命令 python manage.py drive可以启动小车。所以这一部分文档就来分析manage.py drive这个接口,manage.py的另一个接口是python manage.py train用来训练小车。

可先看一下驾驶小车时的入口程序：



其中dk.load\_config()是donkeycar目录下的config.py中的load\_config（）函数。主要作用是从main目录下（即d2下的）config.py获取配置信息，并把配置信息以key-value对的形式设置为cfg的属性。

然后看一下drive函数

## Drive (cfg, model\_path=None, use\_joystick=False)

首先实例化小车：

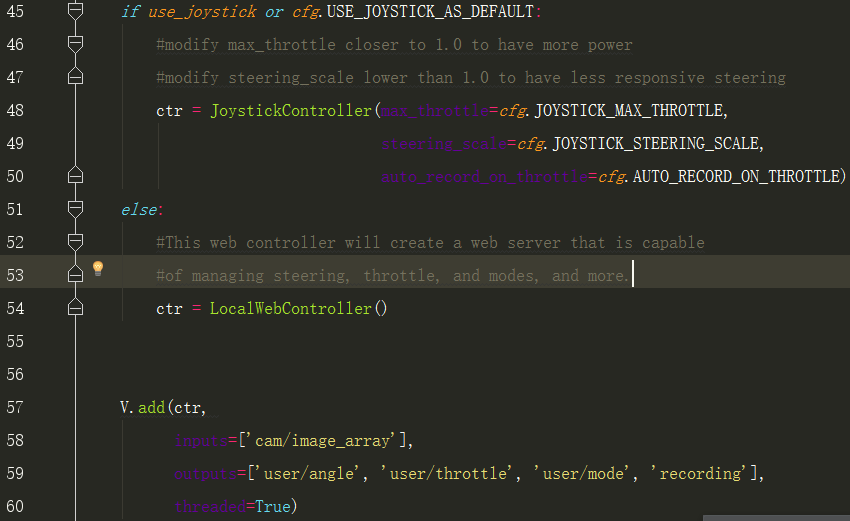


加载摄像头：



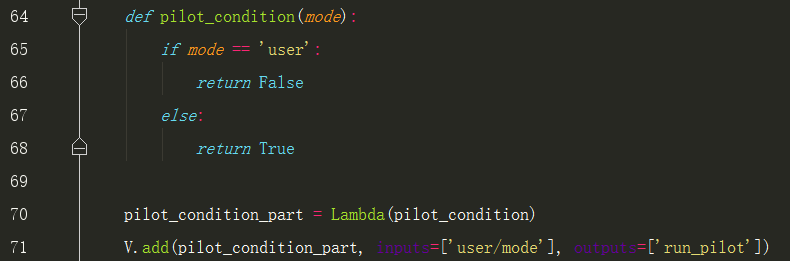
其中PiCamera（）函数来自于引入的包文件donkeycar.parts.camera 具体见【附录1】

确定控制器，并加入小车的循环控制系统：



本案例中使用的LocalWebController()，详细见附录【2】

判断驾驶模式，并将驾驶模式的判断函数封装成小车的一个模块（Lambda()函数实现）：

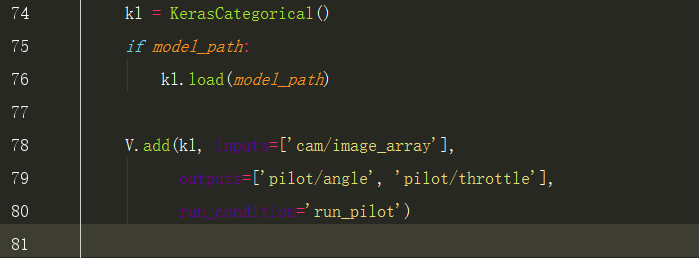


判断是否需要自动驾驶，如果需要，加入自动驾驶模式，可以对比启动自动驾驶的命令

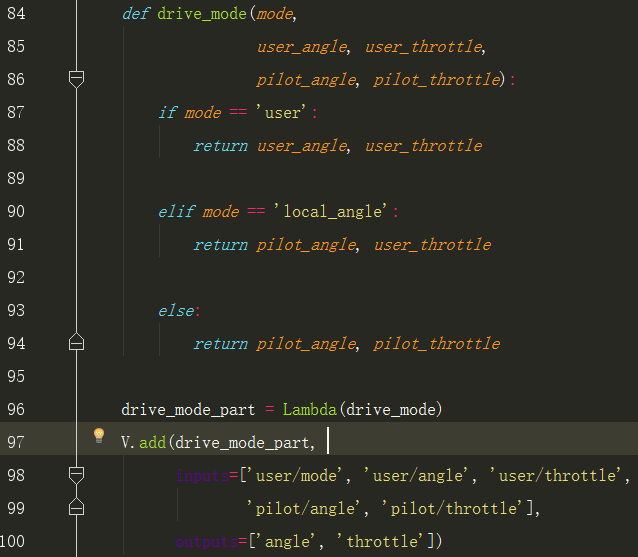
python manage.py drive --model ~/d2/models/mypilot：

注意上述代码中的mode==’user’的mode是其input中的‘user/model’中的mode，也即ctr模块的outputs中的‘user/mode’。

以下是加载自动驾驶模块，如果需要修改模型，就是修改这部分的内容，即kl = KerasCategorical(),关于KerasCategorical()下文会有叙述。见附录【7】



确定驾驶模式，并封装成小车模块（Lambda()函数实现，Lambda函数见附录【3】）：



小车马达控制，这部分之后的都不用改，也不用关心怎么实现的，就是pca的驱动接口。



## train(cfg, tub\_names, model\_name):

利用tub\_name中的数据训练模型并存储在model\_name中，

*def* train(*cfg*, *tub\_names*, *model\_name*):  
 '''  
 use the specified data in tub\_names to train an artifical neural network  
 saves the output trained model as model\_name  
 '''  
 X\_keys = ['cam/image\_array']  
 y\_keys = ['user/angle', 'user/throttle']  
  
 *def* rt(*record*):  
 *record*['user/angle'] = dk.utils.linear\_bin(*record*['user/angle'])  
 *return record* kl = KerasCategorical()  
 *print*('tub\_names', *tub\_names*)  
 *if not tub\_names*:  
 tub\_names = os.path.join(*cfg*.DATA\_PATH, '\*')  
 tubgroup = TubGroup(*tub\_names*)  
 train\_gen, val\_gen = tubgroup.get\_train\_val\_gen(X\_keys, y\_keys, record\_transform=rt,  
 batch\_size=*cfg*.BATCH\_SIZE,  
 train\_frac=*cfg*.TRAIN\_TEST\_SPLIT)  
  
 model\_path = os.path.expanduser(*model\_name*)  
  
 total\_records = len(tubgroup.df)  
 total\_train = int(total\_records \* *cfg*.TRAIN\_TEST\_SPLIT)  
 total\_val = total\_records - total\_train  
 *print*('train: %d, validation: %d' % (total\_train, total\_val))  
 steps\_per\_epoch = total\_train // *cfg*.BATCH\_SIZE  
 *print*('steps\_per\_epoch', steps\_per\_epoch)  
  
 kl.train(train\_gen,  
 val\_gen,  
 saved\_model\_path=model\_path,  
 steps=steps\_per\_epoch,  
 train\_split=*cfg*.TRAIN\_TEST\_SPLIT)

# 引用

1. <https://github.com/wroscoe/donkey/>

# 附录【1】donkeycar.parts.camera

附录1主要介绍donkeycar.parts.camera



# 附录【2】LocalWebController

附录2主要介绍LocalWebController()

LocalWebController的位置在donkeycar.parts.web\_controller.web中，是一个用Tornado框架写的网络服务器，







# 附录【3】Lambda

附录【3】主要介绍将函数封装成小车模块的Lambda类:

|  |
| --- |
| class Lambda:  """  Wraps a function into a donkey part.  """  def \_\_init\_\_(self, f):  """  Accepts the function to use.  """  self.f = f    def run(self, \*args, \*\*kwargs):  return self.f(\*args, \*\*kwargs)    def shutdown(self):  return |

关于\*args, \*\*kwargs，用于当参数类型和参数个数不确定的时候使用，\*args, 是一个元组类型的，\*\*kwargs是字典类型的。

# 附录【4】PCA9685

附录【4】主要介绍类PCA9685，主要用于控制马达

# 附录【5】TubGroup()

附录【5】主要用于介绍TubGroup()，数据存储格式

TubGroup的父类时Tub(),

*def* get\_train\_val\_gen(self, *X\_keys*, *Y\_keys*, *batch\_size*=128, *record\_transform*=None, *train\_frac*=.8):  
 train\_df = train=self.df.sample(frac=*train\_frac*,random\_state=200)  
 val\_df = self.df.drop(train\_df.index)  
  
 train\_gen = self.get\_train\_gen(X\_keys=*X\_keys*, Y\_keys=*Y\_keys*, batch\_size=*batch\_size*,  
 record\_transform=*record\_transform*, df=train\_df)  
  
 val\_gen = self.get\_train\_gen(X\_keys=*X\_keys*, Y\_keys=*Y\_keys*, batch\_size=*batch\_size*,  
 record\_transform=*record\_transform*, df=val\_df)  
  
 *return* train\_gen, val\_gen

|  |
| --- |
| def get\_train\_gen(self, X\_keys, Y\_keys, batch\_size=128, record\_transform=None, df=None):  batch\_gen = self.get\_batch\_gen(X\_keys + Y\_keys,  batch\_size=batch\_size, record\_transform=record\_transform, df=df)  while True:  batch = next(batch\_gen)  X = [batch[k] for k in X\_keys]  Y = [batch[k] for k in Y\_keys]  yield X, Y |

|  |
| --- |
| *def* get\_batch\_gen(self, *keys*, *record\_transform*=None, *batch\_size*=128, *shuffle*=True, *df*=None):   record\_gen = self.get\_record\_gen(*record\_transform*, shuffle=*shuffle*, df=*df*)   *if keys* == None:  keys = list(self.df.columns)   *while* True:  record\_list = []  *for* \_ *in* range(*batch\_size*):  record\_list.append(next(record\_gen))   batch\_arrays = {}  *for* i, k *in* enumerate(*keys*):  arr = np.array([r[k] *for* r *in* record\_list])  # if len(arr.shape) == 1:  # arr = arr.reshape(arr.shape + (1,))  batch\_arrays[k] = arr   *yield* batch\_arrays |

|  |
| --- |
| def get\_record\_gen(self, record\_transform=None, shuffle=True, df=None):  if df is None:  df = self.get\_df()  while True:  for row in self.df.iterrows():  if shuffle:  record\_dict = df.sample(n=1).to\_dict(orient='record')[0]  if record\_transform:  record\_dict = record\_transform(record\_dict)  record\_dict = self.read\_record(record\_dict)  yield record\_dict |

meta.json格式如下：

{  
    **"types"**:[  
        **"image\_array"**,  
        **"float"**,  
        **"float"**,  
        **"str"**  
    ],  
    **"inputs"**:[  
        **"cam/image\_array"**,  
        **"user/angle"**,  
        **"user/throttle"**,  
        **"user/mode"**  
    ]  
}

Record.json格式如下：

{  
    **"cam/image\_array"**:**"1\_cam-image\_array\_.jpg"**,  
    **"user/angle"**:**-8.326672684688674e-17**,  
    **"user/throttle"**:**-3.191891195797325e-16**,  
    **"user/mode"**:**"user"**  
}

# 附录【6】KerasPilot()

附录【6】主要介绍KerasPilot()，机器学习的算法类

*class* KerasCategorical(KerasPilot):  
 *def \_\_init\_\_*(self, *model*=None, *\**args, *\*\**kwargs):  
 super(KerasCategorical, self).\_\_init\_\_(\**args*, \*\**kwargs*)  
 *if model*:  
 self.model = *model  
 else*:  
 self.model = default\_categorical()  
   
 *def* run(self, *img\_arr*):  
 img\_arr = *img\_arr*.reshape((1,) + *img\_arr*.shape)  
 angle\_binned, throttle = self.model.predict(*img\_arr*)  
 #print('throttle', throttle)  
 #angle\_certainty = max(angle\_binned[0])  
 angle\_unbinned = dk.utils.linear\_unbin(angle\_binned)  
 *return* angle\_unbinned, throttle[0][0]

|  |
| --- |
| class KerasPilot():    def load(self, model\_path):  self.model = keras.models.load\_model(model\_path)    def shutdown(self):  pass    def train(self, train\_gen, val\_gen,  saved\_model\_path, epochs=100, steps=100, train\_split=0.8,  verbose=1, min\_delta=.0005, patience=5, use\_early\_stop=True):    """  train\_gen: generator that yields an array of images an array of    """  #checkpoint to save model after each epoch  save\_best = keras.callbacks.ModelCheckpoint(saved\_model\_path,  monitor='val\_loss',  verbose=verbose,  save\_best\_only=True,  mode='min')    #stop training if the validation error stops improving.  early\_stop = keras.callbacks.EarlyStopping(monitor='val\_loss',  min\_delta=min\_delta,  patience=patience,  verbose=verbose,  mode='auto')    callbacks\_list = [save\_best]  if use\_early\_stop:  callbacks\_list.append(early\_stop)    hist = self.model.fit\_generator(  train\_gen,  steps\_per\_epoch=steps,  epochs=epochs,  verbose=1,  validation\_data=val\_gen,  callbacks=callbacks\_list,  validation\_steps=steps\*(1.0 - train\_split))  return hist |

*def* default\_categorical():  
 *from* keras.layers *import* Input, Dense, merge  
 *from* keras.models *import* Model  
 *from* keras.layers *import* Convolution2D, MaxPooling2D, Reshape, BatchNormalization  
 *from* keras.layers *import* Activation, Dropout, Flatten, Dense  
   
 img\_in = Input(shape=(120, 160, 3), name='img\_in') # First layer, input layer, Shape comes from camera.py resolution, RGB  
 x = img\_in  
 x = Convolution2D(24, (5,5), strides=(2,2), activation='relu')(x) # 24 features, 5 pixel x 5 pixel kernel (convolution, feauture) window, 2wx2h stride, relu activation  
 x = Convolution2D(32, (5,5), strides=(2,2), activation='relu')(x) # 32 features, 5px5p kernel window, 2wx2h stride, relu activatiion  
 x = Convolution2D(64, (5,5), strides=(2,2), activation='relu')(x) # 64 features, 5px5p kernal window, 2wx2h stride, relu  
 x = Convolution2D(64, (3,3), strides=(2,2), activation='relu')(x) # 64 features, 3px3p kernal window, 2wx2h stride, relu  
 x = Convolution2D(64, (3,3), strides=(1,1), activation='relu')(x) # 64 features, 3px3p kernal window, 1wx1h stride, relu  
  
 # Possibly add MaxPooling (will make it less sensitive to position in image). Camera angle fixed, so may not to be needed  
  
 x = Flatten(name='flattened')(x) # Flatten to 1D (Fully connected)  
 x = Dense(100, activation='relu')(x) # Classify the data into 100 features, make all negatives 0  
 x = Dropout(.1)(x) # Randomly drop out (turn off) 10% of the neurons (Prevent overfitting)  
 x = Dense(50, activation='relu')(x) # Classify the data into 50 features, make all negatives 0  
 x = Dropout(.1)(x) # Randomly drop out 10% of the neurons (Prevent overfitting)  
 #categorical output of the angle  
 angle\_out = Dense(15, activation='softmax', name='angle\_out')(x) # Connect every input with every output and output 15 hidden units. Use Softmax to give percentage. 15 categories and find best one based off percentage 0.0-1.0  
   
 #continous output of throttle  
 throttle\_out = Dense(1, activation='relu', name='throttle\_out')(x) # Reduce to 1 number, Positive number only  
   
 model = Model(inputs=[img\_in], outputs=[angle\_out, throttle\_out])  
 model.compile(optimizer='adam',  
 loss={'angle\_out': 'categorical\_crossentropy',   
 'throttle\_out': 'mean\_absolute\_error'},  
 loss\_weights={'angle\_out': 0.9, 'throttle\_out': .001})  
  
 *return* model

|  |
| --- |
| def save\_model(model, filepath, overwrite=True, include\_optimizer=True):  """Save a model to a HDF5 file.  Note: Please also see  [How can I install HDF5 or h5py to save my models in Keras?](  /getting-started/faq/  #how-can-i-install-HDF5-or-h5py-to-save-my-models-in-Keras)  in the FAQ for instructions on how to install `h5py`.  The saved model contains:  - the model's configuration (topology)  - the model's weights  - the model's optimizer's state (if any)  Thus the saved model can be reinstantiated in  the exact same state, without any of the code  used for model definition or training.  # Arguments  model: Keras model instance to be saved.  filepath: one of the following:  - string, path where to save the model, or  - h5py.File object where to save the model  overwrite: Whether we should overwrite any existing  model at the target location, or instead  ask the user with a manual prompt.  include\_optimizer: If True, save optimizer's state together.  # Raises  ImportError: if h5py is not available.  """  if h5py is None:  raise ImportError('`save\_model` requires h5py.')  def get\_json\_type(obj):  """Serialize any object to a JSON-serializable structure.  # Arguments  obj: the object to serialize  # Returns  JSON-serializable structure representing `obj`.  # Raises  TypeError: if `obj` cannot be serialized.  """  # if obj is a serializable Keras class instance  # e.g. optimizer, layer  if hasattr(obj, 'get\_config'):  return {'class\_name': obj.\_\_class\_\_.\_\_name\_\_,  'config': obj.get\_config()}  # if obj is any numpy type  if type(obj).\_\_module\_\_ == np.\_\_name\_\_:  if isinstance(obj, np.ndarray):  return {'type': type(obj),  'value': obj.tolist()}  else:  return obj.item()  # misc functions (e.g. loss function)  if callable(obj):  return obj.\_\_name\_\_  # if obj is a python 'type'  if type(obj).\_\_name\_\_ == type.\_\_name\_\_:  return obj.\_\_name\_\_  raise TypeError('Not JSON Serializable:', obj)  from .. import \_\_version\_\_ as keras\_version  if not isinstance(filepath, h5py.File):  # If file exists and should not be overwritten.  if not overwrite and os.path.isfile(filepath):  proceed = ask\_to\_proceed\_with\_overwrite(filepath)  if not proceed:  return  f = h5py.File(filepath, mode='w')  opened\_new\_file = True  else:  f = filepath  opened\_new\_file = False  try:  f.attrs['keras\_version'] = str(keras\_version).encode('utf8')  f.attrs['backend'] = K.backend().encode('utf8')  f.attrs['model\_config'] = json.dumps({  'class\_name': model.\_\_class\_\_.\_\_name\_\_,  'config': model.get\_config()  }, default=get\_json\_type).encode('utf8')  model\_weights\_group = f.create\_group('model\_weights')  model\_layers = model.layers  save\_weights\_to\_hdf5\_group(model\_weights\_group, model\_layers)  if include\_optimizer and model.optimizer:  if isinstance(model.optimizer, optimizers.TFOptimizer):  warnings.warn(  'TensorFlow optimizers do not '  'make it possible to access '  'optimizer attributes or optimizer state '  'after instantiation. '  'As a result, we cannot save the optimizer '  'as part of the model save file.'  'You will have to compile your model again '  'after loading it. '  'Prefer using a Keras optimizer instead '  '(see keras.io/optimizers).')  else:  f.attrs['training\_config'] = json.dumps({  'optimizer\_config': {  'class\_name': model.optimizer.\_\_class\_\_.\_\_name\_\_,  'config': model.optimizer.get\_config()  },  'loss': model.loss,  'metrics': model.metrics,  'sample\_weight\_mode': model.sample\_weight\_mode,  'loss\_weights': model.loss\_weights,  }, default=get\_json\_type).encode('utf8')  # Save optimizer weights.  symbolic\_weights = getattr(model.optimizer, 'weights')  if symbolic\_weights:  optimizer\_weights\_group = f.create\_group(  'optimizer\_weights')  weight\_values = K.batch\_get\_value(symbolic\_weights)  weight\_names = []  for i, (w, val) in enumerate(zip(symbolic\_weights,  weight\_values)):  # Default values of symbolic\_weights is /variable  # for Theano and CNTK  if K.backend() == 'theano' or K.backend() == 'cntk':  if hasattr(w, 'name'):  if w.name.split('/')[-1] == 'variable':  name = str(w.name) + '\_' + str(i)  else:  name = str(w.name)  else:  name = 'param\_' + str(i)  else:  if hasattr(w, 'name') and w.name:  name = str(w.name)  else:  name = 'param\_' + str(i)  weight\_names.append(name.encode('utf8'))  optimizer\_weights\_group.attrs[  'weight\_names'] = weight\_names  for name, val in zip(weight\_names, weight\_values):  param\_dset = optimizer\_weights\_group.create\_dataset(  name,  val.shape,  dtype=val.dtype)  if not val.shape:  # scalar  param\_dset[()] = val  else:  param\_dset[:] = val  f.flush()  finally:  if opened\_new\_file:  f.close() |

# 附录【7】KerasCategorica

附录【7】主要介绍KerasCategorical在小车运行时的输出的信号:

小车运行时，Keras模块的作用主要是根据训练的模型生成控制信号，从manage.py文件的一下代码：

|  |
| --- |
| kl = KerasCategorical() *if model\_path*:  kl.load(*model\_path*)  V.add(kl, inputs=['cam/image\_array'],   outputs=['pilot/angle', 'pilot/throttle'],  run\_condition='run\_pilot') |

|  |
| --- |
| def load(self, model\_path):  self.model = keras.models.load\_model(model\_path) |

可以看出，当有训练好的模型时，小车首先根据位置载入模型，在小车调用start函数的时候将输出信号输出到'pilot/angle', 'pilot/throttle'中，而实现的过程就是该模块的run方法。

|  |
| --- |
| *def* run(self, *img\_arr*):  img\_arr = *img\_arr*.reshape((1,) + *img\_arr*.shape)  angle\_binned, throttle = self.model.predict(*img\_arr*)  #print('throttle', throttle)  #angle\_certainty = max(angle\_binned[0])  angle\_unbinned = dk.utils.linear\_unbin(angle\_binned)  *return* angle\_unbinned, throttle[0][0] |

如上所示，run方法中输出结果的来源主要是model.predict函数，查看Keras Model.predict的源码可以知道，该函数主要的作用就是根据模型和输入信息给出一个预测结果。

# 附录【8】Vehicle()

附录【8】主要介绍一下小车的架构，整体来说，小车提供了一个平台，任何符合接口规

范的模块都可以加载上去，类似于一个支持组件插拔的架构。组织形式类似下图：

黑色部分是小车目前已有的模块，红色部分是如果想要对小车进行扩展可以采取的方案：

编写插件代码，将对应需要交互的数据写进memory，在需要调用的地方直接访问这些数据就行。类似于共享内存区的生产者消费者模型。



以下是UML图



### Vehicle.\_\_init\_\_():

初始化函数，主要声明并初始化了成员变量self.mem, self.parts, self.on, self.threads，

self.parts包含了目前小车所有需要运行的部件

|  |
| --- |
| def \_\_init\_\_(self, mem=None):  if not mem:  mem = Memory()  self.mem = mem  self.parts = []  self.on = True  self.threads = [] |

#### Memory()



### Vehicle.add():

主要是用于将摄像头、驱动单元等部件加载到小车的循环控制单元中，代码中表示主要是将部件信息加载到self.parts中。

参数part, inputs=[], outputs=[], threaded=False, run\_condition=None

part 部件名称

inputs 从内存中获取数据的通道名列表

outputs 往内存中存储数据时的通道名列表

threaded 表示该部分代码是否需要单独的线程来运行

### Vehicle.start():

故名思议，开启小车的循环控制代码。是小车的主要线程，将开启在self.parts中的所有部件以及子线程，然后初始化运行的各个部件并定期刷新内存。

主要参数：rate\_hz=10, max\_loop\_count=None

rate\_hz：整型数,表示控制循环应该运行的最大频率，当有很多障碍是实际的运行速率会比这个低。

max\_loop\_count：整型数，表示控制循环要运行的最大次数，用于测试小车的各个部件是否工作。



### Vehicle. update\_parts ()：

刷新self.parts中的部件信息

主要代码：在for循环中运行

|  |
| --- |
| p = entry['part']  #从内存中获取输入  inputs = self.mem.get(entry['inputs'])  #运行输入的信息  if entry.get('thread'):  outputs = p.run\_threaded(\*inputs)  else:  outputs = p.run(\*inputs)  #将输出结果保存到outputs里  if outputs is not None:  self.mem.put(entry['outputs'], outputs) |

附录【9】