前文ROS机器人底盘(3)-通讯协议定义了PIBOT的通讯协议，该协议较为简单且不与ROS相关同时可以扩展，本文使用python实现一个收发，可以同时使用在windows或者linux

1. dataHolder

python的struct包可以用来打包和解包字节流，针对协议我们定义一个dataHolder包用于打包所有的协议,下面直接上代码

```
import struct

# main board
class MessageID:
    ID_GET_VERSION = 0
    ID_SET_ROBOT_PARAMETER = 1
    ID_GET_ROBOT_PARAMETER = 2
    ID_INIT_ODOM = 3
    ID_SET_VEL = 4
    ID_GET_ODOM = 5
    ID_GET_PID_DEBUG = 6
    ID_GET_IMU = 7

class RobotMessage:
    def pack(self):
        return b''

    def unpack(self):
        return True

class RobotFirmwareInfo(RobotMessage):
    def __init__(self):
        self.version = ''
        self.build_time = ''

    def unpack(self, data):
        try:
            upk = struct.unpack('16s16s', bytes(data))
        except:
            return False
        [self.version, self.build_time] = upk
        return True

class RobotParameters():
    def __init__(self, wheel_diameter=0, \
                 wheel_track=0, \
                 encoder_resolution=0, \
                 do_pid_interval=0, \
                 kp=0, \
                 ki=0, \
                 kd=0, \
                 ko=0, \
```

```
                cmd_last_time=0, \
                max_v_liner_x=0, \
                max_v_liner_y=0, \
                max_v_angular_z=0, \
                imu_type=0, \
                ):
        self.wheel_diameter = wheel_diameter
        self.wheel_track = wheel_track
        self.encoder_resolution = encoder_resolution
        self.do_pid_interval = do_pid_interval
        self.kp = kp
        self.ki = ki
        self.kd = kd
        self.ko = ko
        self.cmd_last_time = cmd_last_time
        self.max_v_liner_x = max_v_liner_x
        self.max_v_liner_y = max_v_liner_y
        self.max_v_angular_z = max_v_angular_z
        self.imu_type = imu_type
        self.reserve = b"01234567890123456789012345678 9"

robotParam = RobotParameters()

class GetRobotParameters(RobotMessage):
    def __init__(self):
        self.param = robotParam

    def unpack(self, data):
        upk = struct.unpack('<3H1B8H1B%ds'%(64-(3*2+1+8*2+1)), bytes(data))

        [self.param.wheel_diameter,
         self.param.wheel_track,
         self.param.encoder_resolution,
         self.param.do_pid_interval,
         self.param.kp,
         self.param.ki,
         self.param.kd,
         self.param.ko,
         self.param.cmd_last_time,
         self.param.max_v_liner_x,
         self.param.max_v_liner_y,
         self.param.max_v_angular_z,
         self.param.imu_type,
         self.param.reserve] = upk
        return True

class SetRobotParameters(RobotMessage):
    def __init__(self):
        self.param = robotParam

    def pack(self):
        data = [self.param.wheel_diameter,
                self.param.wheel_track,
                self.param.encoder_resolution,
```

```
                self.param.do_pid_interval,
                self.param.kp,
                self.param.ki,
                self.param.kd,
                self.param.ko,
                self.param.cmd_last_time,
                self.param.max_v_liner_x,
                self.param.max_v_liner_y,
                self.param.max_v_angular_z,
                self.param.imu_type,
                self.param.reserve]

        pk = struct.pack('<3H1B8H1B%ds'%(64-(3*2+1+8*2+1)), *data)
        return pk

    def unpack(self, data):
        return True

class RobotVel(RobotMessage):
    def __init__(self):
        self.v_liner_x = 0
        self.v_liner_y = 0
        self.v_angular_z = 0

    def pack(self):
        data = [self.v_liner_x,
                self.v_liner_y,
                self.v_angular_z]
        pk = struct.pack('3h', *data)
        return pk

    def unpack(self, data):
        return True

#todo the rest of the message classes
class RobotOdom(RobotMessage):
    def __init__(self):
        self.v_liner_x = 0
        self.v_liner_y = 0
        self.v_angular_z = 0
        self.x = 0
        self.y = 0
        self.yaw = 0

    def unpack(self, data):
        try:
            upk = struct.unpack('<3H2l1H', bytes(data))
        except:
            return False
        [self.v_liner_x,
                self.v_liner_y,
                self.v_angular_z,
                self.x,
                self.y,
```

```
                self.yaw] = upk
        return True

class RobotPIDData(RobotMessage):
    pass

class RobotIMU(RobotMessage):
    def __init__(self):
        self.imu = [0]*9

    def unpack(self, data):
        try:
            upk = struct.unpack('<9f', bytes(data))
        except:
            return False

        self.imu = upk
        return True

BoardDataDict = {MessageID.ID_GET_VERSION:RobotFirmwareInfo(),
            MessageID.ID_GET_ROBOT_PARAMETER:GetRobotParameters(),
            MessageID.ID_SET_ROBOT_PARAMETER:SetRobotParameters(),
            MessageID.ID_SET_VEL:RobotVel(),
            MessageID.ID_GET_ODOM:RobotOdom(),
            MessageID.ID_GET_PID_DEBUG: RobotPIDData(),
            MessageID.ID_GET_IMU: RobotIMU(),
            }
```

- 每条协议定义一个类，该类都从RobotMessage继承而来，针对协议读写类比分别实现pack和unpack接口
- BoardDataDict为id到该类对象的映射绑定

> 新增协议只需要新增实现一个从RobotMessage继承的类，同时添加到BoardDataDict映射表中即可，具体可以参考IMU数据

2 transport

```
import sys
sys.path.append("..")
import pypibot
from pypibot import log

import serial
import threading
import struct
import time
from dataholder import MessageID, BoardDataDict
FIX_HEAD = 0x5a

class Recstate():
```

```
    WAITING_HD = 0
    WAITING_MSG_ID = 1
    RECEIVE_LEN = 2
    RECEIVE_PACKAGE = 3
    RECEIVE_CHECK = 4

def checksum(d):
    sum = 0
    for i in d:
        sum += ord(i)
        sum = sum&0xff
    return sum


class Transport:
    def __init__(self, port, baudrate=921600):
        self._Port = port
        self._Baudrate = baudrate
        self._KeepRunning = False
        self.receive_state = Recstate.WAITING_HD
        self.rev_msg = []
        self.rev_data = []
        self.wait_event = threading.Event()

    def getDataHolder(self):
        return BoardDataDict

    def start(self):
        try:
            self._Serial = serial.Serial(port=self._Port, baudrate=self._Baudrate,
timeout=0.2)
            self._KeepRunning = True
            self._ReceiverThread = threading.Thread(target=self._Listen)
            self._ReceiverThread.setDaemon(True)
            self._ReceiverThread.start()
            return True
        except:
            return False

    def Stop(self):
        self._KeepRunning = False
        time.sleep(0.1)
        self._Serial.close()

    def _Listen(self):
        while self._KeepRunning:
            if self.receiveFiniteStates(self._Serial.read()):
                self.packageAnalysis()

    def receiveFiniteStates(self, s):
        if len(s) == 0:
            return  False
        val = s[0]
        if self.receive_state == Recstate.WAITING_HD:
```

```
            if ord(val) == FIX_HEAD:

                log.debug('got head')
                self.rev_msg = []
                self.rev_data =[]
                self.rev_msg.append(val)
                self.receive_state = Recstate.WAITING_MSG_ID
        elif self.receive_state == Recstate.WAITING_MSG_ID:
            log.debug('got msg id')
            self.rev_msg.append(val)
            self.receive_state = Recstate.RECEIVE_LEN
        elif self.receive_state == Recstate.RECEIVE_LEN:
            log.debug('got len:%d', ord(val))
            self.rev_msg.append(val)
            if ord(val) == 0:
                self.receive_state = Recstate.RECEIVE_CHECK
            else:
                self.receive_state = Recstate.RECEIVE_PACKAGE
        elif self.receive_state == Recstate.RECEIVE_PACKAGE:
            self.rev_data.append(val)
            if len(self.rev_data) == ord(self.rev_msg[-1]):
                self.rev_msg.extend(self.rev_data)
                self.receive_state = Recstate.RECEIVE_CHECK
        elif self.receive_state == Recstate.RECEIVE_CHECK:
            log.debug('got check')
            self.receive_state = Recstate.WAITING_HD
            if ord(val) == checksum(self.rev_msg):
                log.debug('got a complete message')
                return True
        else:
            self.receive_state = Recstate.WAITING_HD

        # continue receiving
        return False

    def packageAnalysis(self):
        in_msg_id = ord(self.rev_msg[1])
        log.debug(bytes(self.rev_data))
        if BoardDataDict[in_msg_id].unpack(''.join(self.rev_data)):
            self.res_id = in_msg_id
            if in_msg_id<100:
                self.set_response()
            else:#notify
                log.debug('msg %d'%self.rev_msg[4],'data incoming')
                pass
        else:
            log.debug ('error unpacking pkg')

    def request(self, id, timeout=0.5):
        if not self.write(id):
            log.debug ('Serial send error!')
            return False
        if self.wait_for_response(timeout):
            if id == self.res_id:
```

```
                log.debug ('OK')
            else:
                log.error ('Got unmatched response!')
        else:
            log.error ('Request got no response!')
            return False
        # clear response
        self.res_id = None
        return True

    def write(self, id):
        self._Serial.write(self.make_command(id))
        return True

    def wait_for_response(self, timeout):
        self.wait_event.clear()
        return self.wait_event.wait(timeout)

    def set_response(self):
        self.wait_event.set()

    def make_command(self, id):
        data = BoardDataDict[id].pack()
        l = [FIX_HEAD, id, len(data)]
        head = struct.pack("3B", *l)
        body = head + data
        return body + chr(checksum(body))

if __name__ == '__main__':
    mboard = Transport('com1')
    if not mboard.start():
        import sys
        sys.exit()

    p = mboard.request(MessageID.ID_GET_VERSION)
    log.i("result=%s"%p)
```

- 利用`serial`包完成串口通讯
- 打开接收线程用来处理回复消息
- 可以看到在发送消息的时，`make_command`中通过`data = BoardDataDict[id].pack()`打包得到数据
- 同事在接收消息的时，`packageAnalysis`中通过`BoardDataDict[in_msg_id].unpack(''.join(self.rev_data)`解包得到数据

测试

`main.py`

```
import platform
import sys
sys.path.append("..")
```

```
import pypibot
from pypibot import log
from transport import Transport
from dataholder import MessageID
import params

port="com10"
TEST_SET_PARAM = True

pypibot.enableGlobalExcept()
#log.enableFileLog(log_dir + "ros_$(Date8)_$(filenumber2).log")
log.setLevel("i")

if __name__ == '__main__':
    mboard = Transport(port, params.pibotBaud)
    if not mboard.start():
        log.error("can not open %s"%port)
        sys.exit()

    DataHolder = mboard.getDataHolder()

    log.info("***************get robot version*****************")
    boardVersion = DataHolder[MessageID.ID_GET_VERSION]
    p = mboard.request(MessageID.ID_GET_VERSION)
    if p:
        log.info("firmware version:%s buildtime:%s\r\n"%
(boardVersion.version.decode(), boardVersion.build_time.decode()))
    else:
        log.error('request firmware version err\r\n')
        quit(1)

    # get robot parameter
    robotParam = DataHolder[MessageID.ID_GET_ROBOT_PARAMETER]
    p = mboard.request(MessageID.ID_GET_ROBOT_PARAMETER)
    if p:
        log.info("wheel_diameter:%d wheel_track:%d encoder_resolution:%d" \
                 %(robotParam.param.wheel_diameter, \
                   robotParam.param.wheel_track, \
                   robotParam.param.encoder_resolution
                   ))

        log.info("do_pid_interval:%d kp:%d ki:%d kd:%d ko:%d" \
                 %(robotParam.param.do_pid_interval, \
                   robotParam.param.kp, \
                   robotParam.param.ki, \
                   robotParam.param.kd, \
                   robotParam.param.ko))

        log.info("cmd_last_time:%d imu_type:%d" \
                 %(robotParam.param.cmd_last_time,\
                   robotParam.param.imu_type
                   ))

        log.info("max_v:%d %d %d\r\n" \
```

```
                %(robotParam.param.max_v_liner_x,\
                  robotParam.param.max_v_liner_y, \
                  robotParam.param.max_v_angular_z
                  ))
    else:
        log.error('request get param err\r\n')
        quit(1)


    if TEST_SET_PARAM:
        log.info("***************set robot parameter***************")

        DataHolder[MessageID.ID_SET_ROBOT_PARAMETER].param = params.pibotParam

        p = mboard.request(MessageID.ID_SET_ROBOT_PARAMETER)
        if p:
            log.info('request set parameter success')
        else:
            log.error('request set parameter err')
            quit(1)

    log.info("***************get odom&imu***************")
    while True:
        robotOdom = DataHolder[MessageID.ID_GET_ODOM]
        p = mboard.request(MessageID.ID_GET_ODOM)
        if p:
            log.info('request get odom success vx=%d vy=%d vangular=%d x=%d y=%d
yaw=%d'%(robotOdom.v_liner_x, \
                                                        robotOdom.v_liner_y, \
                                                        robotOdom.v_angular_z, \
                                                        robotOdom.x, \
                                                        robotOdom.y, \
                                                        robotOdom.yaw))
        else:
            log.error('request get odom err')
            quit(1)

        robotIMU = DataHolder[MessageID.ID_GET_IMU].imu
        p = mboard.request(MessageID.ID_GET_IMU)
        if p:
            log.info('request get imu success \nimu=[%f %f %f\n    %f %f %f\n
%f %f %f]\n'%(robotIMU[0], robotIMU[1], robotIMU[2], \

robotIMU[3], robotIMU[4], robotIMU[5], \

robotIMU[6], robotIMU[7], robotIMU[8]))
        else:
            log.error('request get imu err')
            quit(1)
```