UNIVERSITATEA TITU MAIORESCU

FACULTATEA: INFORMATICĂ
DEPARTAMENT: INFORMATICĂ
Programa de studii: INFORMATICĂ
DISCIPLINA: INTELIGENȚĂ ARTIFICIALĂ

IA – *Testul de evaluare nr. 2*

Robot autonom terestru – FLEXY BOT

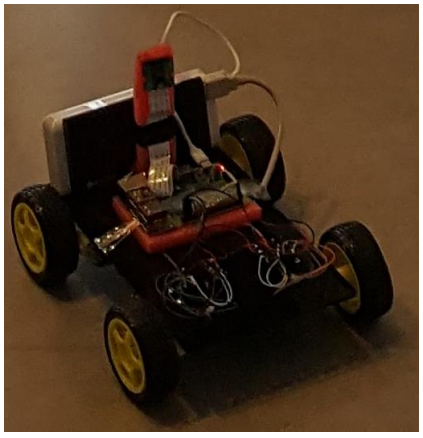
Grupa	Numele și prenumele	Semnătură student	Notă evaluare

Data: ____/____/_____

Conf.dr.ing.
Lucian Ștefăniță GRIGORE

Conf.dr.ing.
Iustin PRIESCU

Ș.L.dr.ing.
Dan-Laurențiu GRECU



Cuprins

1. INTRODUCERE

IA specifică roboților mobili tereștri autonomi trebui să funcționeze după o schemă (simplificată), în care funcția generală a robotului să prmită traversarea unui tern nestructurat, necunoscut și să poată transporta o sarcină utilă. Chiar dacă misiunile principale ale oricărui robot sunt acelea de inspecție, catografiere, filmare și mapare a terenului, el trebuie să poată avea un potențial care se poate upgrada în funcție de misiunile care pot apare.

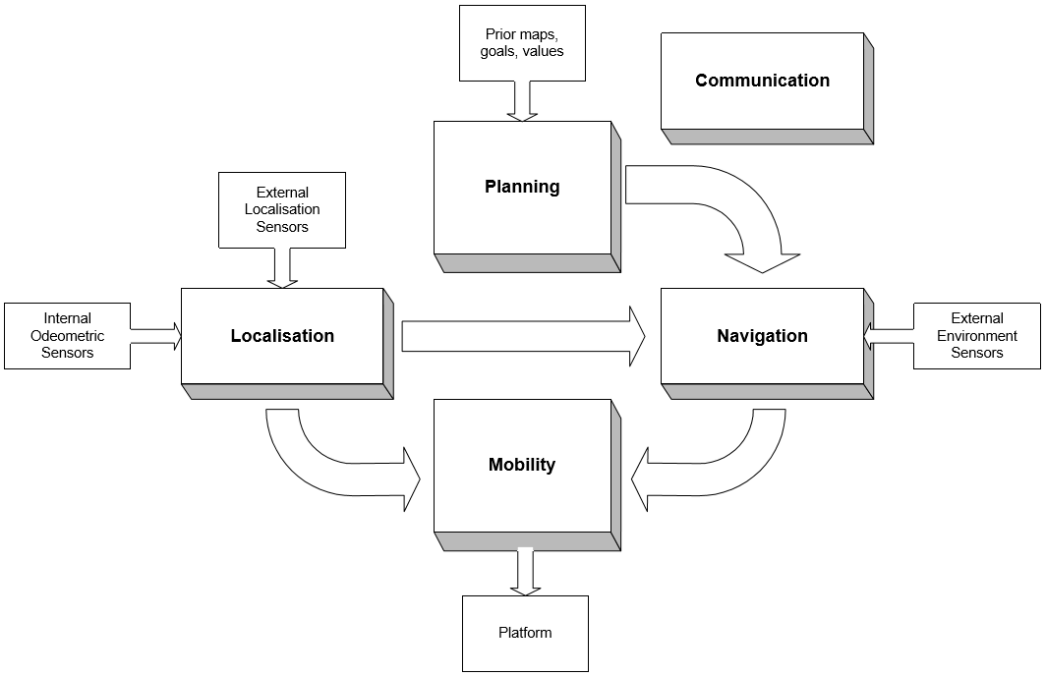


Fig. 1-1 *Schematizarea funcționării robotului Flexy Bot*
https://pdfs.semanticscholar.org/3b9b/0d138378ec2085944f3c77202c6e05e5088f.pdf

Localizarea este o problemă de stabilire a poziției unui vehicul în raport cu un anumit sistem de coordonate. Metodele de localizare autonome, reprezintă o provocare suplimentară în tratarea gradului ridicat de incertitudine a senzorilor, în interpretarea observațiilor despre teren și mediu și a necesității de a lua decizii corecte, în mod autonom, pe baza datelor primite de la senzori. Utilizarea sistemelor de navigație inerțiale (INS) în aplicațiile roboților autonomi serveasc drept bază pentru orice proiectare a sistemului de localizare.

Există o structură general acceptată a majorității algoritmilor de localizare (Fig. 1-2). Este alcătuită din trei părți aranjate într-o configurare de feedback. Senzorii de viteză furnizează estimări integrate privind poziția, atitudinea și viteza vehiculului. Aceste estimări sunt, de asemenea, alocate unui algoritm de estimare care, de asemenea, ia informații de la un set de dispozitive externe de detectare. Algoritmul de estimare produce un set de corecții care sunt readuse la senzorii de viteză. Ieșirea senzorilor de viteză este apoi corectată pentru a reflecta informațiile senzorului extern și pentru a obține o estimare de localizare fuzionată. Această structură cuprinde o serie de principii importante:

Algoritmul de estimare (adesea un filtru Kalman) are o caracteristică de pass-pass. Astfel, măsurătorile senzorilor externi apar la ieșirea structurii. În schimb, amplasarea senzorilor de viteză în buclă asigură că măsurătorile senzorilor de rată au o caracteristică "- 1" sau trecere înaltă. Astfel, ieșirea sistemului de localizare este o informație de înaltă frecvență de la senzorii de viteză împreună cu informații de frecvență redusă de la senzori externi. Aceasta corespunde experienței comune; accelerometrele și alți senzori de viteză oferă măsurători ale mișcării rapide, în timp ce GPS sau alți senzori externi oferă măsurători ale locației lentă sau înclinată. Ambele tipuri de senzori sunt esențiale pentru funcționarea cu succes a unui sistem de localizare.

Structura este astfel încât pierderea de senzori externi sau pierderea corecțiilor din algoritmul de estimare determină sistemul să funcționeze doar pe detectarea ratei. Aceasta este o configurare robustă. În general, senzorii de viteză (cum ar fi dispozitivele inerțiale) sunt interni și nu sunt predispuși la defecțiuni, în

timp ce senzorii externi și algoritmii pot eșua în orice număr de moduri imprevizibile. De asemenea, structura este susceptibilă să includă informații corecte externe variate și sporadice; de la observațiile de pe baliză, de la observațiile de teren etc. Deoarece informațiile externe ajung, se efectuează o corecție a senzorilor de rată.

Structura corectorului de eroare este bazată pe un algoritm de estimare, care funcționează prin compararea erorilor dintre informațiile despre rata de înregistrare date, poziția și caraceristica de funcționare. Utilizarea erorii mai degrabă decât a valorii absolute în algoritmii de estimare permite utilizarea mai generalizată și mai precisă a structurilor de estimare. Ar trebui să fie clar din structura procesului de localizare și dintr-o înțelegere în funcție de frecvență a caracteristicilor de zgomot ale senzorului, că atât senzorii de viteză cât și senzorii absoluți sunt necesari pentru a construi un sistem complet de localizare. Algoritmul de estimare utilizează de regulă un filtru Kalman. Cu toate acestea, algoritmii bayesieni sunt de asemenea utilizați fie sub formă de filtre de particule, fie ca estimatori ai funcției de densitate.

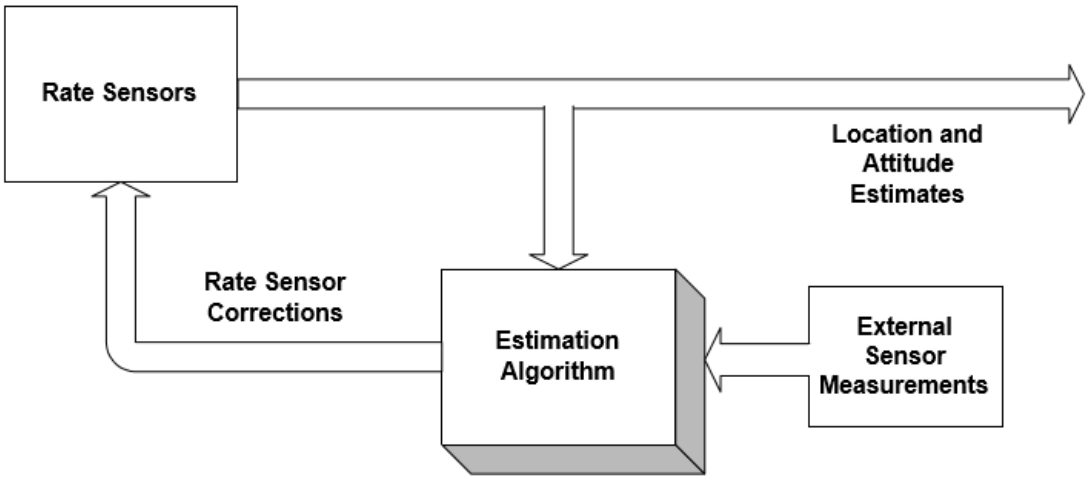


Fig. 1-2 *Schematizarea algoritmului de localizare pentru robotul Flexy Bot*
https://pdfs.semanticscholar.org/3b9b/0d138378ec2085944f3c77202c6e05e5088f.pdf

2. SOFTWARE FLEXYBOT AUTONOM

```
*.pyc
/build
/dist
__pycache__
import wiringpi
# Motor speeds for this library are specified as numbers
# between -MAX_SPEED and MAX_SPEED, inclusive.
_max_speed = 480  # 19.2 MHz / 2 / 480 = 20 kHz
MAX_SPEED = _max_speed
io_initialized = False
def io_init():
  global io_initialized
  if io_initialized:
    return
  wiringpi.wiringPiSetupGpio()
  wiringpi.pinMode(12, wiringpi.GPIO.PWM_OUTPUT)
  wiringpi.pinMode(13, wiringpi.GPIO.PWM_OUTPUT)
  wiringpi.pwmSetMode(wiringpi.GPIO.PWM_MODE_MS)
  wiringpi.pwmSetRange(MAX_SPEED)
  wiringpi.pwmSetClock(2)
  wiringpi.pinMode(5, wiringpi.GPIO.OUTPUT)
  wiringpi.pinMode(6, wiringpi.GPIO.OUTPUT)
  io_initialized = True
class Motor(object):
    MAX_SPEED = _max_speed
    def __init__(self, pwm_pin, dir_pin):
        self.pwm_pin = pwm_pin
        self.dir_pin = dir_pin
    def setSpeed(self, speed):
        if speed < 0:
            speed = -speed
            dir_value = 1
        else:
            dir_value = 0
        if speed > MAX_SPEED:
            speed = MAX_SPEED
        io_init()
        wiringpi.digitalWrite(self.dir_pin, dir_value)
        wiringpi.pwmWrite(self.pwm_pin, speed)
class Motors(object):
    MAX_SPEED = _max_speed
    def __init__(self):
        self.motor1 = Motor(12, 5)
        self.motor2 = Motor(13, 6)
    def setSpeeds(self, m1_speed, m2_speed):
        self.motor1.setSpeed(m1_speed)
        self.motor2.setSpeed(m2_speed)
motors = Motors()
from distutils.core import setup
setup(name='pololu_drv8835_rpi',
      version='2.0.0',
      description=('Library for the Pololu DRV8835 Dual Motor'
                   'Driver Kit for Raspberry Pi Model B+'),
      url='http://www.pololu.com/product/2753',
```

```
      py_modules=['pololu_drv8835_rpi'],
      )
from __future__ import print_function
import time
from pololu_drv8835_rpi import motors, MAX_SPEED
# Set up sequences of motor speeds.
test_forward_speeds = list(range(0, MAX_SPEED, 1)) + \
  [MAX_SPEED] * 200 + list(range(MAX_SPEED, 0, -1)) + [0]
test_reverse_speeds = list(range(0, -MAX_SPEED, -1)) + \
  [-MAX_SPEED] * 200 + list(range(-MAX_SPEED, 0, 1)) + [0]
try:
    motors.setSpeeds(0, 0)
    print("Motor 1 forward")
    for s in test_forward_speeds:
        motors.motor1.setSpeed(s)
        time.sleep(0.005)
    print("Motor 1 reverse")
    for s in test_reverse_speeds:
        motors.motor1.setSpeed(s)
        time.sleep(0.005)
    print("Motor 2 forward")
    for s in test_forward_speeds:
        motors.motor2.setSpeed(s)
        time.sleep(0.005)
    print("Motor 2 reverse")
    for s in test_reverse_speeds:
        motors.motor2.setSpeed(s)
        time.sleep(0.005)
finally:
  # Stop the motors, even if there is an exception
  # or the user presses Ctrl+C to kill the process.
  motors.setSpeeds(0, 0)
build/
*.egg-info/
dist/
__pycache__
*.pyc
wiringpi_wrap.c
wiringpi.py
[submodule "WiringPi"]
path = WiringPi
url = http://github.com/wiringPi/WiringPi
v1.0.0 -- Branched from original WiringPi to deliver new WiringPi 2 functionality
v1.0.1 -- Fixed build problems involving missing header files
v1.0.2 -- Fixed build issue with piNes.c
v1.0.3 -- Fixed bug in physical pin assignment mode
v1.0.4 -- Added class wrapper, plus analogRead/Write functions
v1.0.5 -- Second attempt at pretty Pypi page
v1.0.6 -- Fixed spelling error in softToneCreate - Thanks oevsegneev
v1.0.7 -- Added LCD functionality
v1.0.8 -- Updated manifest to include .rst and fix build error
v1.0.9 -- Erroneous non-fix due to stupidity
v1.0.10 -- Added I2CSetup and new I2C class
v1.1.0 -- Synced to WiringPi as of 8th March 2015
v1.1.1 -- Included devLib folder for headers
v1.2.1 -- Synced to WIringPi as of 27th February 2016
graft WiringPi/wiringPi
```

```
graft WiringPi/devLib
include README.md
include LICENSE.txt
include setup.cfg
include wiringpi.py
include wiringpi_wrap.c
all: bindings
python setup.py build
bindings:
swig3.0 -python -threads wiringpi.i
clean:
rm -rf build/
rm -rf dist/
install:
sudo python setup.py install
// Generated by generate-bindings.py - do not edit manually!
// Header file WiringPi/wiringPi/wiringPi.h
extern int wiringPiFailure (int fatal, const char *message, ...) ;
extern struct wiringPiNodeStruct *wiringPiFindNode (int pin) ;
extern struct wiringPiNodeStruct *wiringPiNewNode  (int pinBase, int numPins) ;
extern int  wiringPiSetup       (void) ;
extern int  wiringPiSetupSys    (void) ;
extern int  wiringPiSetupGpio   (void) ;
extern int  wiringPiSetupPhys   (void) ;
extern void pinModeAlt          (int pin, int mode) ;
extern void pinMode             (int pin, int mode) ;
extern void pullUpDnControl     (int pin, int pud) ;
extern int  digitalRead         (int pin) ;
extern void digitalWrite        (int pin, int value) ;
extern void pwmWrite            (int pin, int value) ;
extern int  analogRead          (int pin) ;
extern void analogWrite         (int pin, int value) ;
extern          int  piBoardRev         (void) ;
extern          void piBoardId          (int *model, int *rev, int *mem, int *maker, int *overVolted) ;
extern          int  wpiPinToGpio       (int wpiPin) ;
extern          int  physPinToGpio      (int physPin) ;
extern          void setPadDrive        (int group, int value) ;
extern          int  getAlt             (int pin) ;
extern          void pwmToneWrite       (int pin, int freq) ;
extern          void digitalWriteByte   (int value) ;
extern unsigned int  digitalReadByte    (void) ;
extern          void pwmSetMode         (int mode) ;
extern          void pwmSetRange        (unsigned int range) ;
extern          void pwmSetClock        (int divisor) ;
extern          void gpioClockSet       (int pin, int freq) ;
extern int  waitForInterrupt    (int pin, int mS) ;
extern int  piThreadCreate      (void *(*fn)(void *)) ;
extern void piLock              (int key) ;
extern void piUnlock            (int key) ;
extern int piHiPri (const int pri) ;
extern void         delay           (unsigned int howLong) ;
extern void         delayMicroseconds (unsigned int howLong) ;
extern unsigned int millis          (void) ;
extern unsigned int micros          (void) ;
// Header file WiringPi/wiringPi/wiringPiI2C.h
extern int wiringPiI2CRead          (int fd) ;
extern int wiringPiI2CReadReg8      (int fd, int reg) ;
```

```
extern int wiringPiI2CReadReg16      (int fd, int reg) ;
extern int wiringPiI2CWrite          (int fd, int data) ;
extern int wiringPiI2CWriteReg8      (int fd, int reg, int data) ;
extern int wiringPiI2CWriteReg16     (int fd, int reg, int data) ;
extern int wiringPiI2CSetupInterface (const char *device, int devId) ;
extern int wiringPiI2CSetup          (const int devId) ;
// Header file WiringPi/wiringPi/wiringPiSPI.h
int wiringPiSPIGetFd     (int channel) ;
int wiringPiSPIDataRW    (int channel, unsigned char *data, int len) ;
int wiringPiSPISetupMode (int channel, int speed, int mode) ;
int wiringPiSPISetup     (int channel, int speed) ;
// Header file WiringPi/wiringPi/wiringSerial.h
extern int   serialOpen      (const char *device, const int baud) ;
extern void  serialClose     (const int fd) ;
extern void  serialFlush     (const int fd) ;
extern void  serialPutchar   (const int fd, const unsigned char c) ;
extern void  serialPuts      (const int fd, const char *s) ;
extern void  serialPrintf    (const int fd, const char *message, ...) ;
extern int   serialDataAvail (const int fd) ;
extern int   serialGetchar   (const int fd) ;
// Header file WiringPi/wiringPi/wiringShift.h
extern uint8_t shiftIn      (uint8_t dPin, uint8_t cPin, uint8_t order) ;
extern void    shiftOut     (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val) ;
// Header file WiringPi/wiringPi/drcSerial.h
extern int drcSetupSerial (const int pinBase, const int numPins, const char *device, const int baud) ;
// Header file WiringPi/wiringPi/ads1115.h
extern int ads1115Setup (int pinBase, int i2cAddress) ;
// Header file WiringPi/wiringPi/max31855.h
extern int max31855Setup (int pinBase, int spiChannel) ;
// Header file WiringPi/wiringPi/max5322.h
extern int max5322Setup (int pinBase, int spiChannel) ;
// Header file WiringPi/wiringPi/mcp23008.h
extern int mcp23008Setup (const int pinBase, const int i2cAddress) ;
// Header file WiringPi/wiringPi/mcp23016.h
extern int mcp23016Setup (const int pinBase, const int i2cAddress) ;
// Header file WiringPi/wiringPi/mcp23016reg.h
// Header file WiringPi/wiringPi/mcp23017.h
extern int mcp23017Setup (const int pinBase, const int i2cAddress) ;
// Header file WiringPi/wiringPi/mcp23s08.h
extern int mcp23s08Setup (const int pinBase, const int spiPort, const int devId) ;
// Header file WiringPi/wiringPi/mcp23s17.h
extern int mcp23s17Setup (int pinBase, int spiPort, int devId) ;
// Header file WiringPi/wiringPi/mcp23x0817.h
// Header file WiringPi/wiringPi/mcp23x08.h
// Header file WiringPi/wiringPi/mcp3002.h
extern int mcp3002Setup (int pinBase, int spiChannel) ;
// Header file WiringPi/wiringPi/mcp3004.h
extern int mcp3004Setup (int pinBase, int spiChannel) ;
// Header file WiringPi/wiringPi/mcp3422.h
extern int mcp3422Setup (int pinBase, int i2cAddress, int sampleRate, int gain) ;
// Header file WiringPi/wiringPi/mcp4802.h
extern int mcp4802Setup (int pinBase, int spiChannel) ;
// Header file WiringPi/wiringPi/pcf8574.h
extern int pcf8574Setup (const int pinBase, const int i2cAddress) ;
// Header file WiringPi/wiringPi/pcf8591.h
extern int pcf8591Setup (const int pinBase, const int i2cAddress) ;
// Header file WiringPi/wiringPi/sn3218.h
```

```
extern int sn3218Setup (int pinBase) ;
// Header file WiringPi/wiringPi/softPwm.h
extern int  softPwmCreate (int pin, int value, int range) ;
extern void softPwmWrite  (int pin, int value) ;
extern void softPwmStop   (int pin) ;
// Header file WiringPi/wiringPi/softServo.h
extern void softServoWrite  (int pin, int value) ;
extern int softServoSetup   (int p0, int p1, int p2, int p3, int p4, int p5, int p6, int p7) ;
// Header file WiringPi/wiringPi/softTone.h
extern int  softToneCreate (int pin) ;
extern void softToneStop   (int pin) ;
extern void softToneWrite  (int pin, int freq) ;
// Header file WiringPi/wiringPi/sr595.h
extern int sr595Setup (const int pinBase, const int numPins,
   const int dataPin, const int clockPin, const int latchPin) ;
// Header file WiringPi/devLib/ds1302.h
extern unsigned int ds1302rtcRead       (const int reg) ;
extern void         ds1302rtcWrite      (const int reg, const unsigned int data) ;
extern unsigned int ds1302ramRead       (const int addr) ;
extern void         ds1302ramWrite      (const int addr, const unsigned int data) ;
extern void         ds1302clockRead     (int clockData [8]) ;
extern void         ds1302clockWrite    (const int clockData [8]) ;
extern void         ds1302trickleCharge (const int diodes, const int resistors) ;
extern void         ds1302setup         (const int clockPin, const int dataPin, const int csPin) ;
// Header file WiringPi/devLib/font.h
// Header file WiringPi/devLib/gertboard.h
extern void gertboardAnalogWrite (const int chan, const int value) ;
extern int  gertboardAnalogRead  (const int chan) ;
extern int  gertboardSPISetup    (void) ;
extern int  gertboardAnalogSetup (const int pinBase) ;
// Header file WiringPi/devLib/lcd128x64.h
extern void lcd128x64setOrigin         (int x, int y) ;
extern void lcd128x64setOrientation    (int orientation) ;
extern void lcd128x64orientCoordinates (int *x, int *y) ;
extern void lcd128x64getScreenSize     (int *x, int *y) ;
extern void lcd128x64point             (int  x, int  y, int colour) ;
extern void lcd128x64line              (int x0, int y0, int x1, int y1, int colour) ;
extern void lcd128x64lineTo            (int  x, int  y, int colour) ;
extern void lcd128x64rectangle         (int x1, int y1, int x2, int y2, int colour, int filled) ;
extern void lcd128x64circle            (int  x, int  y, int  r, int colour, int filled) ;
extern void lcd128x64ellipse           (int cx, int cy, int xRadius, int yRadius, int colour, int filled) ;
extern void lcd128x64putchar           (int  x, int  y, int c, int bgCol, int fgCol) ;
extern void lcd128x64puts              (int  x, int  y, const char *str, int bgCol, int fgCol) ;
extern void lcd128x64update            (void) ;
extern void lcd128x64clear             (int colour) ;
extern int  lcd128x64setup             (void) ;
// Header file WiringPi/devLib/lcd.h
extern void lcdHome        (const int fd) ;
extern void lcdClear       (const int fd) ;
extern void lcdDisplay     (const int fd, int state) ;
extern void lcdCursor      (const int fd, int state) ;
extern void lcdCursorBlink (const int fd, int state) ;
extern void lcdSendCommand (const int fd, unsigned char command) ;
extern void lcdPosition    (const int fd, int x, int y) ;
extern void lcdCharDef     (const int fd, int index, unsigned char data [8]) ;
extern void lcdPutchar     (const int fd, unsigned char data) ;
extern void lcdPuts        (const int fd, const char *string) ;
```

```
extern void lcdPrintf      (const int fd, const char *message, ...) ;
extern int  lcdInit (const int rows, const int cols, const int bits,
   const int rs, const int strb,
   const int d0, const int d1, const int d2, const int d3, const int d4,
   const int d5, const int d6, const int d7) ;
// Header file WiringPi/devLib/maxdetect.h
int maxDetectRead (const int pin, unsigned char buffer [4]) ;
int readRHT03 (const int pin, int *temp, int *rh) ;
// Header file WiringPi/devLib/piGlow.h
extern void piGlow1     (const int leg,  const int ring, const int intensity) ;
extern void piGlowLeg   (const int leg,  const int intensity) ;
extern void piGlowRing  (const int ring, const int intensity) ;
extern void piGlowSetup (int clear) ;
// Header file WiringPi/devLib/piNes.h
extern int          setupNesJoystick (int dPin, int cPin, int lPin) ;
extern unsigned int  readNesJoystick (int joystick) ;
// Header file WiringPi/devLib/scrollPhat.h
extern void scrollPhatPoint      (int x, int y, int colour) ;
extern void scrollPhatLine       (int x0, int y0, int x1, int y1, int colour) ;
extern void scrollPhatLineTo     (int x, int y, int colour) ;
extern void scrollPhatRectangle  (int x1, int y1, int x2, int y2, int colour, int filled) ;
extern void scrollPhatUpdate     (void) ;
extern void scrollPhatClear      (void) ;
extern int  scrollPhatPutchar    (int c) ;
extern void scrollPhatPuts       (const char *str) ;
extern void scrollPhatPrintf     (const char *message, ...) ;
extern void scrollPhatPrintSpeed (const int cps10) ;
extern void scrollPhatIntensity  (const int percent) ;
extern int  scrollPhatSetup      (void) ;
swig2.0 -python -threads wiringpi.i
sudo python setup.py build install
sudo python test.py
%pythoncode %{
# wiringPi modes
WPI_MODE_PINS = 0;
WPI_MODE_GPIO = 1;
WPI_MODE_GPIO_SYS = 2;
WPI_MODE_PHYS = 3;
WPI_MODE_PIFACE = 4;
WPI_MODE_UNINITIALISED = -1;
# Pin modes
INPUT = 0;
OUTPUT = 1;
PWM_OUTPUT = 2;
GPIO_CLOCK = 3;
SOFT_PWM_OUTPUT = 4;
SOFT_TONE_OUTPUT = 5;
PWM_TONE_OUTPUT = 6;
LOW = 0;
HIGH = 1;
# Pull up/down/none
PUD_OFF = 0;
PUD_DOWN = 1;
PUD_UP = 2;
# PWM
PWM_MODE_MS = 0;
PWM_MODE_BAL = 1;
```

```
# Interrupt levels
INT_EDGE_SETUP = 0;
INT_EDGE_FALLING = 1;
INT_EDGE_RISING = 2;
INT_EDGE_BOTH = 3;
%}
HEADERS = []
src = open("wiringpi.i").read().split('\n')
for line in src:
    line = line.strip()
    if line.startswith('#include') and line.endswith('.h"'):
        HEADERS.append(line.replace('#include','').replace('"','').strip())
#print(HEADERS)
def is_c_decl(line):
    for fn in ['wiringPiISR','wiringPiSetupPiFace','wiringPiSetupPiFaceForGpioProg']:
        if fn in line:
            return False
    for prefix in ['extern','void','int','uint8_t']:
        if line.startswith(prefix):
            return True
print("// Generated by generate-bindings.py - do not edit manually!")
for file in HEADERS:
    print("\n// Header file {}".format(file))
    h = open(file).read().split('\n')
    extern = False
    cont = False
    if 'extern "C" {' not in h:
        extern = True
    for line in h:
        line = line.strip()
        if cont:
            print("\t{}".format(line))
            cont = ";" not in line
            continue
        if line.startswith('extern "C"'):
            extern = True
            continue
        if is_c_decl(line) and extern:
            print(line)
            cont = ";" not in line
[metadata]
description-file = README.md
#!/usr/bin/env python
from setuptools import setup, find_packages, Extension
from glob import glob
sources = glob('WiringPi/devLib/*.c')
sources += glob('WiringPi/wiringPi/*.c')
sources += ['wiringpi_wrap.c']
sources.remove('WiringPi/devLib/piFaceOld.c')
_wiringpi = Extension(
    '_wiringpi',
    include_dirs=['WiringPi/wiringPi','WiringPi/devLib'],
    sources=sources
)
setup(
    name = 'wiringpi',
    version = '2.32.1',
```

```
    author = "Philip Howard",
    author_email = "phil@gadgetoid.com",
    url = 'https://github.com/WiringPi/WiringPi-Python/',
    description = """A python interface to WiringPi 2.0 library which allows for
    easily interfacing with the GPIO pins of the Raspberry Pi. Also supports
    i2c and SPI""",
    long_description=open('README.md').read(),
    ext_modules = [ _wiringpi ],
    py_modules = ["wiringpi"],
    install_requires=[],
    headers=glob('WiringPi/wiringPi/*.h')+glob('WiringPi/devLib/*.h')
)
%pythoncode %{
class nes(object):
  def setupNesJoystick(self,*args):
    return setupNesJoystick(*args)
  def readNesJoystick(self,*args):
    return readNesJoystick(*args)
class Serial(object):
  device = '/dev/ttyAMA0'
  baud = 9600
  serial_id = 0
  def printf(self,*args):
    return serialPrintf(self.serial_id,*args)
  def dataAvail(self,*args):
    return serialDataAvail(self.serial_id,*args)
  def getchar(self,*args):
    return serialGetchar(self.serial_id,*args)
  def putchar(self,*args):
    return serialPutchar(self.serial_id,*args)
  def puts(self,*args):
    return serialPuts(self.serial_id,*args)
  def __init__(self,device,baud):
    self.device = device
    self.baud = baud
    self.serial_id = serialOpen(self.device,self.baud)
  def __del__(self):
    serialClose(self.serial_id)
class I2C(object):
  def setupInterface(self,*args):
   return wiringPiI2CSetupInterface(*args)
  def setup(self,*args):
    return wiringPiI2CSetup(*args)
  def read(self,*args):
    return wiringPiI2CRead(*args)
  def readReg8(self,*args):
    return wiringPiI2CReadReg8(*args)
  def readReg16(self,*args):
    return wiringPiI2CReadReg16(*args)
  def write(self,*args):
    return wiringPiI2CWrite(*args)
  def writeReg8(self,*args):
    return wiringPiI2CWriteReg8(*args)
  def writeReg16(self,*args):
    return wiringPiI2CWriteReg16(*args)
class GPIO(object):
  WPI_MODE_PINS = 0
```

```
WPI_MODE_GPIO = 1
WPI_MODE_GPIO_SYS = 2
WPI_MODE_PHYS = 3
WPI_MODE_PIFACE = 4
WPI_MODE_UNINITIALISED = -1
INPUT = 0
OUTPUT = 1
PWM_OUTPUT = 2
GPIO_CLOCK = 3
LOW = 0
HIGH = 1
PUD_OFF = 0
PUD_DOWN = 1
PUD_UP = 2
PWM_MODE_MS = 0
PWM_MODE_BAL = 1
INT_EDGE_SETUP = 0
INT_EDGE_FALLING = 1
INT_EDGE_RISING = 2
INT_EDGE_BOTH = 3
LSBFIRST = 0
MSBFIRST = 1
MODE = 0
def __init__(self,pinmode=0):
  self.MODE=pinmode
  if pinmode==self.WPI_MODE_PINS:
    wiringPiSetup()
  if pinmode==self.WPI_MODE_GPIO:
    wiringPiSetupGpio()
  if pinmode==self.WPI_MODE_GPIO_SYS:
    wiringPiSetupSys()
  if pinmode==self.WPI_MODE_PHYS:
    wiringPiSetupPhys()
  if pinmode==self.WPI_MODE_PIFACE:
    wiringPiSetupPiFace()
def delay(self,*args):
  delay(*args)
def delayMicroseconds(self,*args):
  delayMicroseconds(*args)
def millis(self):
  return millis()
def micros(self):
  return micros()
def piHiPri(self,*args):
  return piHiPri(*args)
def piBoardRev(self):
  return piBoardRev()
def wpiPinToGpio(self,*args):
  return wpiPinToGpio(*args)
def setPadDrive(self,*args):
  return setPadDrive(*args)
def getAlt(self,*args):
  return getAlt(*args)
def digitalWriteByte(self,*args):
  return digitalWriteByte(*args)
def pwmSetMode(self,*args):
  pwmSetMode(*args)
```

```
def pwmSetRange(self,*args):
  pwmSetRange(*args)
def pwmSetClock(self,*args):
  pwmSetClock(*args)
def gpioClockSet(self,*args):
  gpioClockSet(*args)
def pwmWrite(self,*args):
  pwmWrite(*args)
def pinMode(self,*args):
  pinMode(*args)
def digitalWrite(self,*args):
  digitalWrite(*args)
def digitalRead(self,*args):
  return digitalRead(*args)
def digitalWriteByte(self,*args):
  digitalWriteByte(*args)
def analogWrite(self,*args):
  analogWrite(*args)
def analogRead(self,*args):
  return analogRead(*args)
def shiftOut(self,*args):
  shiftOut(*args)
def shiftIn(self,*args):
  return shiftIn(*args)
def pullUpDnControl(self,*args):
  return pullUpDnControl(*args)
def waitForInterrupt(self,*args):
  return waitForInterrupt(*args)
def wiringPiISR(self,*args):
  return wiringPiISR(*args)
def softPwmCreate(self,*args):
  return softPwmCreate(*args)
def softPwmWrite(self,*args):
  return sofPwmWrite(*args)
def softToneCreate(self,*args):
  return softToneCreate(*args)
def softToneWrite(self,*args):
  return softToneWrite(*args)
def lcdHome(self,*args):
  return lcdHome(self,*args)
def lcdCLear(self,*args):
  return lcdClear(self,*args)
def lcdSendCommand(self,*args):
  return lcdSendCommand(self,*args)
def lcdPosition(self,*args):
  return lcdPosition(self,*args)
def lcdPutchar(self,*args):
  return lcdPutchar(self,*args)
def lcdPuts(self,*args):
  return lcdPuts(self,*args)
def lcdPrintf(self,*args):
  return lcdPrintf(self,*args)
def lcdInit(self,*args):
  return lcdInit(self,*args)
def piGlowSetup(self,*args):
  return piGlowSetup(self,*args)
def piGlow1(self,*args):
```

```
    return piGlow1(self,*args)
  def piGlowLeg(self,*args):
    return piGlowLeg(self,*args)
  def piGlowRing(self,*args):
    return piGlowRing(self,*args)
%}
%module wiringpi
%{
#if PY_MAJOR_VERSION >= 3
#define PyInt_AS_LONG PyLong_AsLong
#define PyString_FromStringAndSize PyBytes_FromStringAndSize
#endif
#include "WiringPi/wiringPi/wiringPi.h"
#include "WiringPi/wiringPi/wiringPiI2C.h"
#include "WiringPi/wiringPi/wiringPiSPI.h"
#include "WiringPi/wiringPi/wiringSerial.h"
#include "WiringPi/wiringPi/wiringShift.h"
#include "WiringPi/wiringPi/drcSerial.h"
#include "WiringPi/wiringPi/ads1115.h"
#include "WiringPi/wiringPi/max31855.h"
#include "WiringPi/wiringPi/max5322.h"
#include "WiringPi/wiringPi/mcp23008.h"
#include "WiringPi/wiringPi/mcp23016.h"
#include "WiringPi/wiringPi/mcp23016reg.h"
#include "WiringPi/wiringPi/mcp23017.h"
#include "WiringPi/wiringPi/mcp23s08.h"
#include "WiringPi/wiringPi/mcp23s17.h"
#include "WiringPi/wiringPi/mcp23x0817.h"
#include "WiringPi/wiringPi/mcp23x08.h"
#include "WiringPi/wiringPi/mcp3002.h"
#include "WiringPi/wiringPi/mcp3004.h"
#include "WiringPi/wiringPi/mcp3422.h"
#include "WiringPi/wiringPi/mcp4802.h"
#include "WiringPi/wiringPi/pcf8574.h"
#include "WiringPi/wiringPi/pcf8591.h"
#include "WiringPi/wiringPi/sn3218.h"
#include "WiringPi/wiringPi/softPwm.h"
#include "WiringPi/wiringPi/softServo.h"
#include "WiringPi/wiringPi/softTone.h"
#include "WiringPi/wiringPi/sr595.h"
#include "WiringPi/devLib/ds1302.h"
#include "WiringPi/devLib/font.h"
#include "WiringPi/devLib/gertboard.h"
#include "WiringPi/devLib/lcd128x64.h"
#include "WiringPi/devLib/lcd.h"
#include "WiringPi/devLib/maxdetect.h"
#include "WiringPi/devLib/piGlow.h"
#include "WiringPi/devLib/piNes.h"
#include "WiringPi/devLib/scrollPhat.h"
%}
%apply unsigned char { uint8_t };
%typemap(in) (unsigned char *data, int len) {
    $1 = (unsigned char *) PyString_AsString($input);
    $2 = PyString_Size($input);
};
// Grab a Python function object as a Python object.
%typemap(in) PyObject *pyfunc {
```

```
  if (!PyCallable_Check($1)) {
      PyErr_SetString(PyExc_TypeError, "Need a callable object!");
      return NULL;
  }
  $1 = $2;
}
%{
// we need to have our own callbacks array
PyObject* event_callback[64] = {0,};
void _wiringPiISR_callback(int pinNumber) {
  PyObject *result;
  if (event_callback[pinNumber]) {
      // this will acquire the GIL
      SWIG_PYTHON_THREAD_BEGIN_BLOCK;
      result = PyObject_CallFunction(event_callback[pinNumber], NULL);
      if (result == NULL && PyErr_Occurred()) {
          PyErr_Print();
          PyErr_Clear();
      }
      Py_XDECREF(result);
      // release the GIL
      SWIG_PYTHON_THREAD_END_BLOCK;
  }
}
/* This is embarrasing, WiringPi does not support supplying args to the callback
... so we have to create callback function for each of the pins :( */
void _wiringPiISR_callback_pin0(void) { _wiringPiISR_callback(0); }
void _wiringPiISR_callback_pin1(void) { _wiringPiISR_callback(1); }
void _wiringPiISR_callback_pin2(void) { _wiringPiISR_callback(2); }
void _wiringPiISR_callback_pin3(void) { _wiringPiISR_callback(3); }
void _wiringPiISR_callback_pin4(void) { _wiringPiISR_callback(4); }
void _wiringPiISR_callback_pin5(void) { _wiringPiISR_callback(5); }
void _wiringPiISR_callback_pin6(void) { _wiringPiISR_callback(6); }
void _wiringPiISR_callback_pin7(void) { _wiringPiISR_callback(7); }
void _wiringPiISR_callback_pin8(void) { _wiringPiISR_callback(8); }
void _wiringPiISR_callback_pin9(void) { _wiringPiISR_callback(9); }
void _wiringPiISR_callback_pin10(void) { _wiringPiISR_callback(10); }
void _wiringPiISR_callback_pin11(void) { _wiringPiISR_callback(11); }
void _wiringPiISR_callback_pin12(void) { _wiringPiISR_callback(12); }
void _wiringPiISR_callback_pin13(void) { _wiringPiISR_callback(13); }
void _wiringPiISR_callback_pin14(void) { _wiringPiISR_callback(14); }
void _wiringPiISR_callback_pin15(void) { _wiringPiISR_callback(15); }
void _wiringPiISR_callback_pin16(void) { _wiringPiISR_callback(16); }
void _wiringPiISR_callback_pin17(void) { _wiringPiISR_callback(17); }
void _wiringPiISR_callback_pin18(void) { _wiringPiISR_callback(18); }
void _wiringPiISR_callback_pin19(void) { _wiringPiISR_callback(19); }
void _wiringPiISR_callback_pin20(void) { _wiringPiISR_callback(20); }
void _wiringPiISR_callback_pin21(void) { _wiringPiISR_callback(21); }
void _wiringPiISR_callback_pin22(void) { _wiringPiISR_callback(22); }
void _wiringPiISR_callback_pin23(void) { _wiringPiISR_callback(23); }
void _wiringPiISR_callback_pin24(void) { _wiringPiISR_callback(24); }
void _wiringPiISR_callback_pin25(void) { _wiringPiISR_callback(25); }
void _wiringPiISR_callback_pin26(void) { _wiringPiISR_callback(26); }
void _wiringPiISR_callback_pin27(void) { _wiringPiISR_callback(27); }
void _wiringPiISR_callback_pin28(void) { _wiringPiISR_callback(28); }
void _wiringPiISR_callback_pin29(void) { _wiringPiISR_callback(29); }
void _wiringPiISR_callback_pin30(void) { _wiringPiISR_callback(30); }
```

```
void _wiringPiISR_callback_pin31(void) { _wiringPiISR_callback(31); }
void _wiringPiISR_callback_pin32(void) { _wiringPiISR_callback(32); }
void _wiringPiISR_callback_pin33(void) { _wiringPiISR_callback(33); }
void _wiringPiISR_callback_pin34(void) { _wiringPiISR_callback(34); }
void _wiringPiISR_callback_pin35(void) { _wiringPiISR_callback(35); }
void _wiringPiISR_callback_pin36(void) { _wiringPiISR_callback(36); }
void _wiringPiISR_callback_pin37(void) { _wiringPiISR_callback(37); }
void _wiringPiISR_callback_pin38(void) { _wiringPiISR_callback(38); }
void _wiringPiISR_callback_pin39(void) { _wiringPiISR_callback(39); }
void _wiringPiISR_callback_pin40(void) { _wiringPiISR_callback(40); }
void _wiringPiISR_callback_pin41(void) { _wiringPiISR_callback(41); }
void _wiringPiISR_callback_pin42(void) { _wiringPiISR_callback(42); }
void _wiringPiISR_callback_pin43(void) { _wiringPiISR_callback(43); }
void _wiringPiISR_callback_pin44(void) { _wiringPiISR_callback(44); }
void _wiringPiISR_callback_pin45(void) { _wiringPiISR_callback(45); }
void _wiringPiISR_callback_pin46(void) { _wiringPiISR_callback(46); }
void _wiringPiISR_callback_pin47(void) { _wiringPiISR_callback(47); }
void _wiringPiISR_callback_pin48(void) { _wiringPiISR_callback(48); }
void _wiringPiISR_callback_pin49(void) { _wiringPiISR_callback(49); }
void _wiringPiISR_callback_pin50(void) { _wiringPiISR_callback(50); }
void _wiringPiISR_callback_pin51(void) { _wiringPiISR_callback(51); }
void _wiringPiISR_callback_pin52(void) { _wiringPiISR_callback(52); }
void _wiringPiISR_callback_pin53(void) { _wiringPiISR_callback(53); }
void _wiringPiISR_callback_pin54(void) { _wiringPiISR_callback(54); }
void _wiringPiISR_callback_pin55(void) { _wiringPiISR_callback(55); }
void _wiringPiISR_callback_pin56(void) { _wiringPiISR_callback(56); }
void _wiringPiISR_callback_pin57(void) { _wiringPiISR_callback(57); }
void _wiringPiISR_callback_pin58(void) { _wiringPiISR_callback(58); }
void _wiringPiISR_callback_pin59(void) { _wiringPiISR_callback(59); }
void _wiringPiISR_callback_pin60(void) { _wiringPiISR_callback(60); }
void _wiringPiISR_callback_pin61(void) { _wiringPiISR_callback(61); }
void _wiringPiISR_callback_pin62(void) { _wiringPiISR_callback(62); }
void _wiringPiISR_callback_pin63(void) { _wiringPiISR_callback(63); }
/* This function adds a new Python function object as a callback object */
static void wiringPiISRWrapper(int pin, int mode, PyObject *PyFunc) {
  // remove the old callback if any
  if (event_callback[pin]) {
    Py_XDECREF(event_callback[pin]);
  }
  // put new callback function
  event_callback[pin] = PyFunc;
  Py_INCREF(PyFunc);
  // and now the ugly switch
  void (*func)(void);
  switch(pin) {
    case 0: func = &_wiringPiISR_callback_pin0; break;
    case 1: func = &_wiringPiISR_callback_pin1; break;
    case 2: func = &_wiringPiISR_callback_pin2; break;
    case 3: func = &_wiringPiISR_callback_pin3; break;
    case 4: func = &_wiringPiISR_callback_pin4; break;
    case 5: func = &_wiringPiISR_callback_pin5; break;
    case 6: func = &_wiringPiISR_callback_pin6; break;
    case 7: func = &_wiringPiISR_callback_pin7; break;
    case 8: func = &_wiringPiISR_callback_pin8; break;
    case 9: func = &_wiringPiISR_callback_pin9; break;
    case 10: func = &_wiringPiISR_callback_pin10; break;
    case 11: func = &_wiringPiISR_callback_pin11; break;
```

```
    case 12: func = &_wiringPiISR_callback_pin12; break;
    case 13: func = &_wiringPiISR_callback_pin13; break;
    case 14: func = &_wiringPiISR_callback_pin14; break;
    case 15: func = &_wiringPiISR_callback_pin15; break;
    case 16: func = &_wiringPiISR_callback_pin16; break;
    case 17: func = &_wiringPiISR_callback_pin17; break;
    case 18: func = &_wiringPiISR_callback_pin18; break;
    case 19: func = &_wiringPiISR_callback_pin19; break;
    case 20: func = &_wiringPiISR_callback_pin20; break;
    case 21: func = &_wiringPiISR_callback_pin21; break;
    case 22: func = &_wiringPiISR_callback_pin22; break;
    case 23: func = &_wiringPiISR_callback_pin23; break;
    case 24: func = &_wiringPiISR_callback_pin24; break;
    case 25: func = &_wiringPiISR_callback_pin25; break;
    case 26: func = &_wiringPiISR_callback_pin26; break;
    case 27: func = &_wiringPiISR_callback_pin27; break;
    case 28: func = &_wiringPiISR_callback_pin28; break;
    case 29: func = &_wiringPiISR_callback_pin29; break;
    case 30: func = &_wiringPiISR_callback_pin30; break;
    case 31: func = &_wiringPiISR_callback_pin31; break;
    case 32: func = &_wiringPiISR_callback_pin32; break;
    case 33: func = &_wiringPiISR_callback_pin33; break;
    case 34: func = &_wiringPiISR_callback_pin34; break;
    case 35: func = &_wiringPiISR_callback_pin35; break;
    case 36: func = &_wiringPiISR_callback_pin36; break;
    case 37: func = &_wiringPiISR_callback_pin37; break;
    case 38: func = &_wiringPiISR_callback_pin38; break;
    case 39: func = &_wiringPiISR_callback_pin39; break;
    case 40: func = &_wiringPiISR_callback_pin40; break;
    case 41: func = &_wiringPiISR_callback_pin41; break;
    case 42: func = &_wiringPiISR_callback_pin42; break;
    case 43: func = &_wiringPiISR_callback_pin43; break;
    case 44: func = &_wiringPiISR_callback_pin44; break;
    case 45: func = &_wiringPiISR_callback_pin45; break;
    case 46: func = &_wiringPiISR_callback_pin46; break;
    case 47: func = &_wiringPiISR_callback_pin47; break;
    case 48: func = &_wiringPiISR_callback_pin48; break;
    case 49: func = &_wiringPiISR_callback_pin49; break;
    case 50: func = &_wiringPiISR_callback_pin50; break;
    case 51: func = &_wiringPiISR_callback_pin51; break;
    case 52: func = &_wiringPiISR_callback_pin52; break;
    case 53: func = &_wiringPiISR_callback_pin53; break;
    case 54: func = &_wiringPiISR_callback_pin54; break;
    case 55: func = &_wiringPiISR_callback_pin55; break;
    case 56: func = &_wiringPiISR_callback_pin56; break;
    case 57: func = &_wiringPiISR_callback_pin57; break;
    case 58: func = &_wiringPiISR_callback_pin58; break;
    case 59: func = &_wiringPiISR_callback_pin59; break;
    case 60: func = &_wiringPiISR_callback_pin60; break;
    case 61: func = &_wiringPiISR_callback_pin61; break;
    case 62: func = &_wiringPiISR_callback_pin62; break;
    case 63: func = &_wiringPiISR_callback_pin63; break;
  }
  // register our dedicated function in WiringPi
  wiringPiISR(pin, mode, func);
}
%}
```

```
// overlay normal function with our wrapper
%rename("wiringPiISR") wiringPiISRWrapper        (int pin, int mode, PyObject *PyFunc);
static void wiringPiISRWrapper(int pin, int mode, PyObject *PyFunc);
%typemap(in) unsigned char data [8] {
  /* Check if is a list */
  if (PyList_Check($input)) {
if(PyList_Size($input) != 8){
    PyErr_SetString(PyExc_TypeError,"must contain 8 items");
    return NULL;
}
    int i = 0;
    $1 = (unsigned char *) malloc(8);
    for (i = 0; i < 8; i++) {
      PyObject *o = PyList_GetItem($input,i);
      if        (PyInt_Check(o)        &&        PyInt_AsLong(PyList_GetItem($input,i))        <=        255        &&
PyInt_AsLong(PyList_GetItem($input,i)) >= 0)
$1[i] = PyInt_AsLong(PyList_GetItem($input,i));
      else {
PyErr_SetString(PyExc_TypeError,"list must contain integers 0-255");
return NULL;
      }
    }
  } else {
    PyErr_SetString(PyExc_TypeError,"not a list");
    return NULL;
  }
};
%typemap(freearg) unsigned char data [8] {
  free((unsigned char *) $1);
}
%typemap(in) (unsigned char *data, int len) {
      $1 = (unsigned char *) PyString_AsString($input);
      $2 = PyString_Size($input);
};
%typemap(argout) (unsigned char *data) {
      $result = SWIG_Python_AppendOutput($result, PyString_FromStringAndSize((char *) $1, result));
};
%include "bindings.i"
%include "constants.py"
%include "wiringpi-class.py"
```

3. SOFTWARE Camera de vedere EO – Raspberry Pi Camera V2.1

În următoarele tabele sunt detaliate metodele PiCamera care utilizează clasele de codificatoare și ce metodă le cer pentru a construi aceste Encodere:

Method(s)	Calls	Returns
capture() capture_continuous()capture_sequence()	_get_image_encoder()	PiCookedOneImageEncoderPiRawOneImageEncoder
capture_sequence()	_get_images_encoder()	PiCookedMultiImageEncoderPiRawMultiImageEncoder
start_recording()record_sequence()	_get_video_encoder()	PiCookedVideoEncoderPiRawVideoEncoder

Este recomandat ca funcția specifică a acestor clase să fie corespunzătoare cerințelor utilizatorului. Pentru a extinde clasa PiCookedVideoEncoder și pentru a stoca câte „*i - cadre*” și a captura câte „*p - cadre*”, se execută programul:

```
import picamera
import picamera.mmal as mmal

# Override PiVideoEncoder to keep track of the number of each type of frame
class MyEncoder(picamera.PiCookedVideoEncoder):
    def start(self, output, motion_output=None):
        self.parent.i_frames = 0
        self.parent.p_frames = 0
        super(MyEncoder, self).start(output, motion_output)

    def _callback_write(self, buf):
        # Only count when buffer indicates it's the end of a frame, and
        # it's not an SPS/PPS header (..._CONFIG)
        if (
                (buf.flags & mmal.MMAL_BUFFER_HEADER_FLAG_FRAME_END) and
                not (buf.flags & mmal.MMAL_BUFFER_HEADER_FLAG_CONFIG)
            ):
            if buf.flags & mmal.MMAL_BUFFER_HEADER_FLAG_KEYFRAME:
                self.parent.i_frames += 1
            else:
                self.parent.p_frames += 1
        # Remember to return the result of the parent method!
        return super(MyEncoder, self)._callback_write(buf)

# Override PiCamera to use our custom encoder for video recording
class MyCamera(picamera.PiCamera):
    def __init__(self):
        super(MyCamera, self).__init__()
        self.i_frames = 0
        self.p_frames = 0

    def _get_video_encoder(
```

```
            self, camera_port, output_port, format, resize, **options):
        return MyEncoder(
            self, camera_port, output_port, format, resize, **options)

with MyCamera() as camera:
    camera.start_recording('foo.h264')
    camera.wait_recording(10)
    camera.stop_recording()
    print('Recording contains %d I-frames and %d P-frames' % (
        camera.i_frames, camera.p_frames))
```

Prelevarea datelor brute de la Bayer (un mozaic filtru Bayer este o matrice de filtre de culoare (CFA) pentru aranjarea filtrelor de culoare RGB pe o rețea pătrată de foto senzori, aranjamentul său particular de filtre de culoare este folosit în majoritatea senzorilor de imagine digitali cu un singur chip, utilizați în camere digitale, camere video și scanere pentru a crea o imagine color, modelul de filtrare este de 50% verde, 25% roșu și 25% albastru, prin urmare se numește și BGGR, RGBG, GRGB sau RGGB.

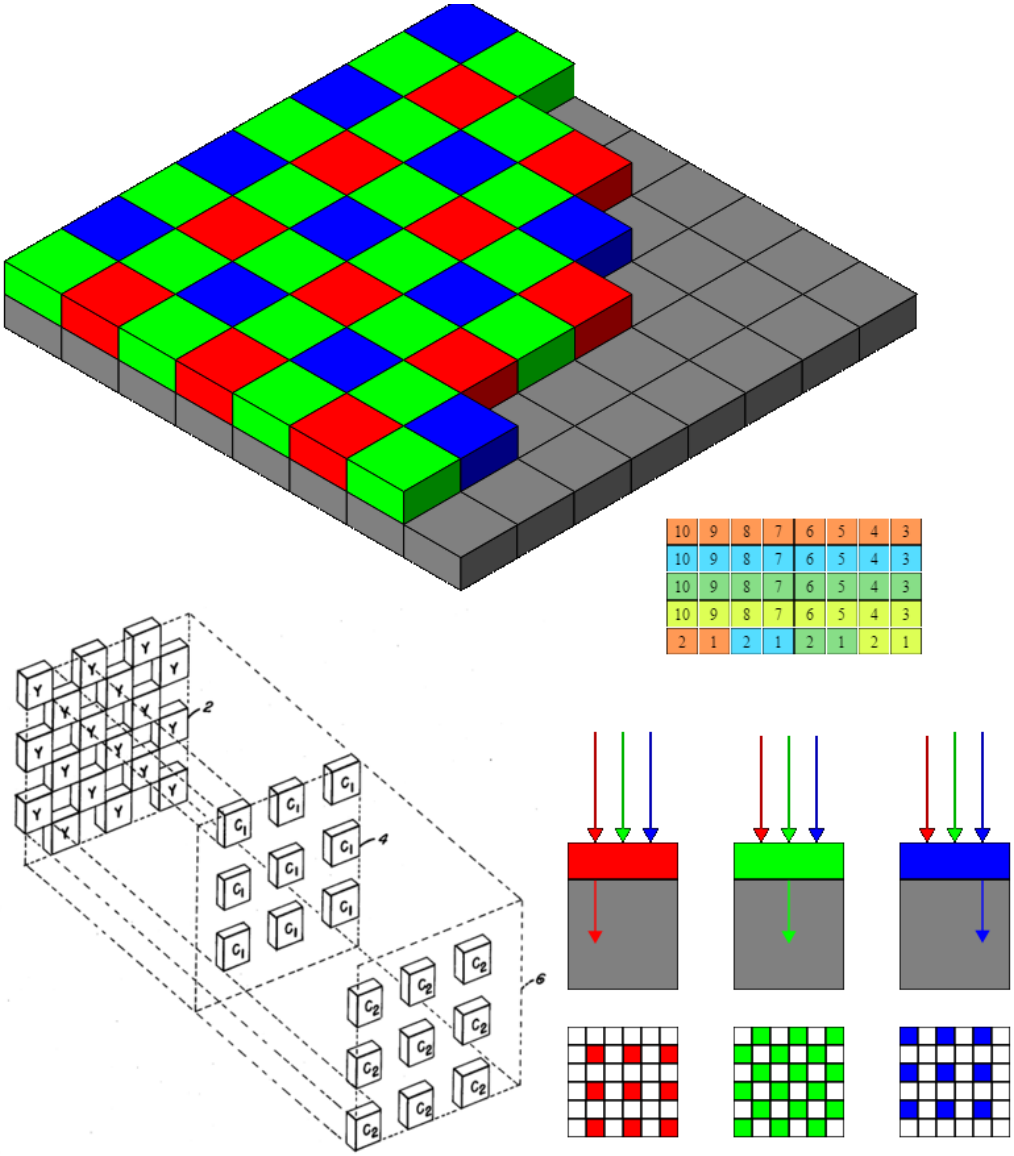


Fig. 3-1 Aranjamentul Bayer al filtrelor de culoare pe matricea pixelilor unui senzor de imagine. http://picamera.readthedocs.io/en/release-1.13/recipes2.html#custom-encoders

Este numit după inventatorul său, Bryce Bayer de la Eastman Kodak. Bayer este de asemenea cunoscut pentru matricea sa definită recursivă utilizată în dithering-ul ordonat.

Alternativele la filtrul Bayer includ atât modificări diferite ale culorilor și aranjamentului, cât și tehnologii complet diferite, cum ar fi eșantionarea în co-site-ul culorilor, senzorul Foveon X3, oglinzile dichroice sau o matrice transparentă a filtrului de difracție.)

Parametrul bayer al metodei capture (Fig. 3-1) determină ca datele brute de la Bayer înregistrate de senzorul camerei să fie trimise ca parte a meta datelor de imagine.

Parametrul bayer funcționează numai cu formatul JPEG și numai pentru capturile de pe port. Datele brute ale Bayer diferă considerabil de capturările simple necodificate; Datele date de senzorul camerei sunt înregistrate înainte de orice procesare grafică a procesorului, incluzând balansul de alb automat, compensarea vignetului, netezirea, scalarea în jos.

PiCamera captează o imagine care include datele brute ale filtrului Bayer. Apoi continuă să despacheteze datele Bayer într-o matrice tridimensională, reprezentând datele RGB brute și în cele din urmă efectuează un pas de-mozaic rudimentar cu medii ponderate, dar trebuie avut în vedere că toate prelucrările se întâmplă pe CPU și vor fi considerabil mai lente decât imaginile obișnuite:

```
from __future__ import (
    unicode_literals,
    absolute_import,
    print_function,
    division,
    )


import io
import time
import picamera
import numpy as np
from numpy.lib.stride_tricks import as_strided

stream = io.BytesIO()
with picamera.PiCamera() as camera:
    # Let the camera warm up for a couple of seconds
    time.sleep(2)
    # Capture the image, including the Bayer data
    camera.capture(stream, format='jpeg', bayer=True)
    ver = {
        'RP_ov5647': 1,
        'RP_imx219': 2,
        }[camera.exif_tags['IFD0.Model']]

# Extract the raw Bayer data from the end of the stream, check the
# header and strip if off before converting the data into a numpy array

offset = {
    1: 6404096,
    2: 10270208,
    }[ver]
data = stream.getvalue()[-offset:]
assert data[:4] == 'BRCM'
data = data[32768:]
data = np.fromstring(data, dtype=np.uint8)

# For the V1 module, the data consists of 1952 rows of 3264 bytes of data.
# The last 8 rows of data are unused (they only exist because the maximum
```

```
# resolution of 1944 rows is rounded up to the nearest 16).
#
# For the V2 module, the data consists of 2480 rows of 4128 bytes of data.
# There's actually 2464 rows of data, but the sensor's raw size is 2466
# rows, rounded up to the nearest multiple of 16: 2480.
#
# Likewise, the last few bytes of each row are unused (why?). Here we
# reshape the data and strip off the unused bytes.

reshape, crop = {
    1: ((1952, 3264), (1944, 3240)),
    2: ((2480, 4128), (2464, 4100)),
    }[ver]
data = data.reshape(reshape)[:crop[0], :crop[1]]

# Horizontally, each row consists of 10-bit values. Every four bytes are
# the high 8-bits of four values, and the 5th byte contains the packed low
# 2-bits of the preceding four values. In other words, the bits of the
# values A, B, C, D and arranged like so:
#
#  byte 1   byte 2   byte 3   byte 4   byte 5
# AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD AABBCCDD
#
# Here, we convert our data into a 16-bit array, shift all values left by
# 2-bits and unpack the low-order bits from every 5th byte in each row,
# then remove the columns containing the packed bits

data = data.astype(np.uint16) << 2
for byte in range(4):
    data[:, byte::5] |= ((data[:, 4::5] >> ((4 - byte) * 2)) & 0b11)
data = np.delete(data, np.s_[4::5], 1)

# Now to split the data up into its red, green, and blue components. The
# Bayer pattern of the OV5647 sensor is BGGR. In other words the first
# row contains alternating green/blue elements, the second row contains
# alternating red/green elements, and so on as illustrated below:
#
# GBGBGBGBGBGBGB
# RGRGRGRGRGRGRG
# GBGBGBGBGBGBGB
# RGRGRGRGRGRGRG
#
# Please note that if you use vflip or hflip to change the orientation
# of the capture, you must flip the Bayer pattern accordingly

rgb = np.zeros(data.shape + (3,), dtype=data.dtype)
rgb[1::2, 0::2, 0] = data[1::2, 0::2] # Red
rgb[0::2, 0::2, 1] = data[0::2, 0::2] # Green
rgb[1::2, 1::2, 1] = data[1::2, 1::2] # Green
rgb[0::2, 1::2, 2] = data[0::2, 1::2] # Blue

# At this point we now have the raw Bayer data with the correct values
# and colors but the data still requires de-mosaicing and
# post-processing. If you wish to do this yourself, end the script here!
#
# Below we present a fairly naive de-mosaic method that simply
# calculates the weighted average of a pixel based on the pixels
```

```
# surrounding it. The weighting is provided by a byte representation of
# the Bayer filter which we construct first:

bayer = np.zeros(rgb.shape, dtype=np.uint8)
bayer[1::2, 0::2, 0] = 1 # Red
bayer[0::2, 0::2, 1] = 1 # Green
bayer[1::2, 1::2, 1] = 1 # Green
bayer[0::2, 1::2, 2] = 1 # Blue

# Allocate an array to hold our output with the same shape as the input
# data. After this we define the size of window that will be used to
# calculate each weighted average (3x3). Then we pad out the rgb and
# bayer arrays, adding blank pixels at their edges to compensate for the
# size of the window when calculating averages for edge pixels.

output = np.empty(rgb.shape, dtype=rgb.dtype)
window = (3, 3)
borders = (window[0] - 1, window[1] - 1)
border = (borders[0] // 2, borders[1] // 2)

rgb = np.pad(rgb, [
    (border[0], border[0]),
    (border[1], border[1]),
    (0, 0),
    ], 'constant')
bayer = np.pad(bayer, [
    (border[0], border[0]),
    (border[1], border[1]),
    (0, 0),
    ], 'constant')

# For each plane in the RGB data, we use a nifty numpy trick
# (as_strided) to construct a view over the plane of 3x3 matrices. We do
# the same for the bayer array, then use Einstein summation on each
# (np.sum is simpler, but copies the data so it's slower), and divide
# the results to get our weighted average:

for plane in range(3):
    p = rgb[..., plane]
    b = bayer[..., plane]
    pview = as_strided(p, shape=(
        p.shape[0] - borders[0],
        p.shape[1] - borders[1]) + window, strides=p.strides * 2)
    bview = as_strided(b, shape=(
        b.shape[0] - borders[0],
        b.shape[1] - borders[1]) + window, strides=b.strides * 2)
    psum = np.einsum('ijkl->ij', pview)
    bsum = np.einsum('ijkl->ij', bview)
    output[..., plane] = psum // bsum

# At this point output should contain a reasonably "normal" looking
# image, although it still won't look as good as the camera's normal
# output (as it lacks vignette compensation, AWB, etc).
#
# If you want to view this in most packages (like GIMP) you'll need to
# convert it to 8-bit RGB data. The simplest way to do this is by
# right-shifting everything by 2-bits (yes, this makes all that
```

```
# unpacking work at the start rather redundant...)

output = (output >> 2).astype(np.uint8)
with open('image.data', 'wb') as f:
    output.tofile(f)
```

4. SOFTWARE Monitor Touchscreen de 7″ pentru Raspberry Pi B V2.1

Toate celelalte suprapuneri periferice care utilizează pini GPIO și se găsesc în situația de conflict, trebuie dezactivate. În config.txt, se introduc comentariile aferente sau se inversează orice dtparams care permit I2C sau SPI:

```
dtparam=i2c_arm=off
dtparam=spi=off
```

Mode	RGB bits	GPIO																							
		27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	565	-	-	-	-	-	-	-	-	7	6	5	4	3	7	6	5	4	3	2	7	6	5	4	3
3	565	-	-	-	7	6	5	4	3	-	-	7	6	5	4	3	2	-	-	-	7	6	5	4	3
4	565	-	-	7	6	5	4	3	-	-	-	7	6	5	4	3	2	-	-	7	6	5	4	3	-
5	666	-	-	-	-	-	-	7	6	5	4	3	2	7	6	5	4	3	2	7	6	5	4	3	2
6	666	-	-	7	6	5	4	3	2	-	-	7	6	5	4	3	2	-	-	7	6	5	4	3	2
7	888	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Controlul parametrilor de ieșire, cum ar fi: ceas, culoare, polaritate, sincronizare, activare, etc. poate fi controlat cu un număr trecut la parametrul dpi_output_format în config.txt creat din următoarele câmpuri:

```
output_format        = (dpi_output_format >>  0) & 0xf;
rgb_order            = (dpi_output_format >>  4) & 0xf;

output_enable_mode   = (dpi_output_format >>  8) & 0x1;
invert_pixel_clock   = (dpi_output_format >>  9) & 0x1;

hsync_disable        = (dpi_output_format >> 12) & 0x1;
vsync_disable        = (dpi_output_format >> 13) & 0x1;
output_enable_disable = (dpi_output_format >> 14) & 0x1;

hsync_polarity       = (dpi_output_format >> 16) & 0x1;
vsync_polarity       = (dpi_output_format >> 17) & 0x1;
output_enable_polarity = (dpi_output_format >> 18) & 0x1;

hsync_phase          = (dpi_output_format >> 20) & 0x1;
vsync_phase          = (dpi_output_format >> 21) & 0x1;
output_enable_phase  = (dpi_output_format >> 22) & 0x1;

output_format:
  1: DPI_OUTPUT_FORMAT_9BIT_666
  2: DPI_OUTPUT_FORMAT_16BIT_565_CFG1
  3: DPI_OUTPUT_FORMAT_16BIT_565_CFG2
  4: DPI_OUTPUT_FORMAT_16BIT_565_CFG3
  5: DPI_OUTPUT_FORMAT_18BIT_666_CFG1
  6: DPI_OUTPUT_FORMAT_18BIT_666_CFG2
  7: DPI_OUTPUT_FORMAT_24BIT_888

rgb_order:
  1: DPI_RGB_ORDER_RGB
  2: DPI_RGB_ORDER_BGR
  3: DPI_RGB_ORDER_GRB
  4: DPI_RGB_ORDER_BRG
```

output_enable_mode:
0: DPI_OUTPUT_ENABLE_MODE_DATA_VALID
1: DPI_OUTPUT_ENABLE_MODE_COMBINED_SYNCS

invert_pixel_clock:
0: RGB Data changes on rising edge and is stable at falling edge
1: RGB Data changes on falling edge and is stable at rising edge.

hsync/vsync/output_enable_polarity:
0: default for HDMI mode
1: inverted

hsync/vsync/oe phases:
0: DPI_PHASE_POSEDGE
1: DPI_PHASE_NEGEDGE

Controlul timpilor sau rezoluțiilor presupune ca parametrii dpi_group și dpi_mode config.txt să fie setați în moduri predeterminate (modurile DMT sau CEA utilizate de HDMI), , sau fiecare utilizator să genereze modul propriu, personalizat. Configurarea în modul HDMI folosește (în config.txt), următoarele instrucțiuni:

dpi_group = 2
dpi_mode = 87

Aceste instrucțiuni vor folosi cronometrele HDMI pentru DPI. Dacă se utilizează opțiunea, parametrul hdmi_timings config.txt este folosit pentru a seta temporizările HDMI (DPI) direct. Parametrii hdmi_timings sunt specificați ca un set de parametri delimitați de spațiu:

hdmi_timings=<h_active_pixels> <h_sync_polarity> <h_front_porch> <h_sync_pulse> <h_back_porch> <v_active_lines> <v_sync_polarity> <v_front_porch> <v_sync_pulse> <v_back_porch> <v_sync_offset_a> <v_sync_offset_b> <pixel_rep> <frame_rate> <interlaced> <pixel_freq> <aspect_ratio>

<h_active_pixels> = horizontal pixels (width)
<h_sync_polarity> = invert hsync polarity
<h_front_porch> = horizontal forward padding from DE acitve edge
<h_sync_pulse> = hsync pulse width in pixel clocks
<h_back_porch> = vertical back padding from DE active edge
<v_active_lines> = vertical pixels height (lines)
<v_sync_polarity> = invert vsync polarity
<v_front_porch> = vertical forward padding from DE active edge
<v_sync_pulse> = vsync pulse width in pixel clocks
<v_back_porch> = vertical back padding from DE active edge
<v_sync_offset_a> = leave at zero
<v_sync_offset_b> = leave at zero
<pixel_rep> = leave at zero
<frame_rate> = screen refresh rate in Hz
<interlaced> = leave at zero
<pixel_freq> = clock frequency (width*height*framerate)
<aspect_ratio> = *

HDMI_ASPECT_4_3 = 1
HDMI_ASPECT_14_9 = 2
HDMI_ASPECT_16_9 = 3
HDMI_ASPECT_5_4 = 4
HDMI_ASPECT_16_10 = 5
HDMI_ASPECT_15_9 = 6
HDMI_ASPECT_21_9 = 7
HDMI_ASPECT_64_27 = 8

```
/dts-v1/;
/ {
   videocore {
          clock_routing {
            vco@PLLD { freq = <2000000000>; };
            chan@DPER { div = <8>; }; // APER will be 500MHz
          };
          pins_rev1 {
            pin_config {
              pin@default {
                polarity = "active_high";
                termination = "pull_down";
                startup_state = "inactive";
                function = "input";
              }; // pin
              pin@p2  { function = "i2c1";   termination = "pull_up"; }; // I2C 1 SDA
              pin@p3  { function = "i2c1";   termination = "pull_up"; }; // I2C 1 SCL
              pin@p5  { function = "output"; termination = "pull_down"; }; // CAM_LED
              pin@p6  { function = "output"; termination = "pull_down"; }; // LAN NRESET
              pin@p14 { function = "uart0";  termination = "no_pulling"; drive_strength_mA = < 8 >; }; // TX uart0
              pin@p15 { function = "uart0";  termination = "pull_up"; drive_strength_mA = < 8 >; }; // RX uart0
              pin@p16 { function = "output"; termination = "pull_up"; polarity="active_low"; }; // activity LED
              pin@p27 { function = "output"; termination = "no_pulling";    }; // Camera shutdown
              pin@p40 { function = "pwm";    termination = "no_pulling"; drive_strength_mA = < 16 >; }; // Left audio
              pin@p45 { function = "pwm";    termination = "no_pulling"; drive_strength_mA = < 16 >; }; // Right audio
              pin@p46 { function = "input";  termination = "no_pulling";    }; // Hotplug
              pin@p47 { function = "input";  termination = "no_pulling";    }; // SD card detect
              pin@p48 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD CLK
              pin@p49 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD CMD
              pin@p50 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D0
              pin@p51 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D1
              pin@p52 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D2
              pin@p53 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D3
            }; // pin_config
            pin_defines {
              pin_define@HDMI_CONTROL_ATTACHED {
                type = "internal";
                number = <46>;
              };
              pin_define@NUM_CAMERAS {
                type = "internal";
                number = <1>;
              };
              pin_define@CAMERA_0_UNICAM_PORT {
                type = "internal";
                number = <1>;
              };
              pin_define@CAMERA_0_I2C_PORT {
                type = "internal";
                number = <1>;
              };
              pin_define@CAMERA_0_SDA_PIN {
                type = "internal";
                number = <2>;
              };
```

```
      pin_define@CAMERA_0_SCL_PIN {
        type = "internal";
        number = <3>;
      };
      pin_define@CAMERA_0_SHUTDOWN {
        type = "internal";
        number = <27>;
      };
      pin_define@CAMERA_0_LED {
        type = "internal";
        number = <5>;
      };
      pin_define@FLASH_0_ENABLE {
        type = "absent";
      };
      pin_define@FLASH_0_INDICATOR {
        type = "absent";
      };
      pin_define@FLASH_1_ENABLE {
        type = "absent";
      };
      pin_define@FLASH_1_INDICATOR {
        type = "absent";
      };
      pin_define@POWER_LOW {
        type = "absent";
      };
      pin_define@LEDS_DISK_ACTIVITY {
        type = "internal";
        number = <16>;
      };
      pin_define@LAN_RESET {
        type = "internal";
        number = <6>;
      };
    }; // pin_defines
  }; // pins_rev1
  pins_rev2 {
    pin_config {
      pin@default {
        polarity = "active_high";
        termination = "pull_down";
        startup_state = "inactive";
        function = "input";
      }; // pin
      pin@p0  { function = "i2c0";   termination = "pull_up"; }; // I2C 0 SDA
      pin@p1  { function = "i2c0";   termination = "pull_up"; }; // I2C 0 SCL
      pin@p5  { function = "output"; termination = "pull_down"; }; // CAM_LED
      pin@p6  { function = "output"; termination = "pull_down"; }; // LAN NRESET
      pin@p14 { function = "uart0";  termination = "no_pulling"; drive_strength_mA = < 8 >; }; // TX uart0
      pin@p15 { function = "uart0";  termination = "pull_up"; drive_strength_mA = < 8 >; }; // RX uart0
      pin@p16 { function = "output"; termination = "pull_up"; polarity = "active_low"; }; // activity LED
      pin@p21 { function = "output"; termination = "no_pulling";    }; // Camera shutdown
      pin@p40 { function = "pwm";    termination = "no_pulling"; drive_strength_mA = < 16 >; }; // Left audio
      pin@p45 { function = "pwm";    termination = "no_pulling"; drive_strength_mA = < 16 >; }; // Right audio
      pin@p46 { function = "input";  termination = "no_pulling";    }; // Hotplug
      pin@p47 { function = "input";  termination = "no_pulling";    }; // SD card detect
```

```
      pin@p48 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD CLK
      pin@p49 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD CMD
      pin@p50 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D0
      pin@p51 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D1
      pin@p52 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D2
      pin@p53 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D3
    }; // pin_config
    pin_defines {
      pin_define@HDMI_CONTROL_ATTACHED {
        type = "internal";
        number = <46>;
      };
      pin_define@NUM_CAMERAS {
        type = "internal";
        number = <1>;
      };
      pin_define@CAMERA_0_I2C_PORT {
        type = "internal";
        number = <0>;
      };
      pin_define@CAMERA_0_SDA_PIN {
        type = "internal";
        number = <0>;
      };
      pin_define@CAMERA_0_SCL_PIN {
        type = "internal";
        number = <1>;
      };
      pin_define@CAMERA_0_SHUTDOWN {
        type = "internal";
        number = <21>;
      };
      pin_define@CAMERA_0_UNICAM_PORT {
        type = "internal";
        number = <1>;
      };
      pin_define@CAMERA_0_LED {
        type = "internal";
        number = <5>;
      };
      pin_define@FLASH_0_ENABLE {
        type = "absent";
      };
      pin_define@FLASH_0_INDICATOR {
        type = "absent";
      };
      pin_define@FLASH_1_ENABLE {
        type = "absent";
      };
      pin_define@FLASH_1_INDICATOR {
        type = "absent";
      };
      pin_define@POWER_LOW {
        type = "absent";
      };
      pin_define@LEDS_DISK_ACTIVITY {
        type = "internal";
```

```
                number = <16>;
              };
              pin_define@LAN_RESET {
                type = "internal";
                number = <6>;
              };
            }; // pin_defines
          }; // pins
          pins_bplus {
            pin_config {
              pin@default {
                polarity = "active_high";
                termination = "pull_down";
                startup_state = "inactive";
                function = "input";
              }; // pin
              pin@p2  { function = "dpi";  termination = "no_pulling"; drive_strength_mA = < 8 >; };
              pin@p3  { function = "dpi";  termination = "no_pulling";    };
              pin@p4  { function = "dpi";  termination = "no_pulling";    };
              pin@p5  { function = "dpi";  termination = "no_pulling";    };
              pin@p6  { function = "dpi";  termination = "no_pulling";    };
              pin@p7  { function = "dpi";  termination = "no_pulling";    };
              pin@p8  { function = "dpi";  termination = "no_pulling";    };
              pin@p9  { function = "dpi";  termination = "no_pulling";    };
              pin@p10 { function = "dpi";  termination = "no_pulling";    };
              pin@p11 { function = "dpi";  termination = "no_pulling";    };
              pin@p12 { function = "dpi";  termination = "no_pulling";    };
              pin@p13 { function = "dpi";  termination = "no_pulling";    };
              pin@p14 { function = "dpi";  termination = "no_pulling";    };
              pin@p15 { function = "dpi";  termination = "no_pulling";    };
              pin@p16 { function = "dpi";  termination = "no_pulling";    };
              pin@p17 { function = "dpi";  termination = "no_pulling";    };
              pin@p18 { function = "dpi";  termination = "no_pulling";    };
              pin@p19 { function = "dpi";  termination = "no_pulling";    };
              pin@p20 { function = "dpi";  termination = "no_pulling";    };
              pin@p21 { function = "dpi";  termination = "no_pulling";    };
              pin@p28 { function = "i2c0";   termination = "pull_up";    }; // I2C 0 SDA
              pin@p29 { function = "i2c0";   termination = "pull_up";    }; // I2C 0 SCL
              pin@p31 { function = "output"; termination = "pull_down"; }; // LAN NRESET
              pin@p32 { function = "output"; termination = "pull_down"; }; // Camera LED
              pin@p35 { function = "input";  termination = "no_pulling"; polarity = "active_low"; }; // Power low
              pin@p38 { function = "output"; termination = "no_pulling";      }; // USB current limit (0=600mA,
1=1200mA)
              pin@p40 { function = "pwm";    termination = "no_pulling"; drive_strength_mA = < 16 >; }; // Left audio
              pin@p41 { function = "output"; termination = "no_pulling";    }; // Camera enable
              pin@p44 { function = "gp_clk"; termination = "pull_down"; }; // Ethernet 25MHz output
              pin@p45 { function = "pwm";    termination = "no_pulling"; drive_strength_mA = < 16 >; }; // Right audio
              pin@p46 { function = "input";  termination = "no_pulling"; polarity = "active_low"; }; // Hotplug
              pin@p47 { function = "output"; termination = "pull_down"; }; // activity LED
              pin@p48 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD CLK
              pin@p49 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD CMD
              pin@p50 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D0
              pin@p51 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D1
              pin@p52 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D2
              pin@p53 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D3
            }; // pin_config
            pin_defines {
```

```
    pin_define@HDMI_CONTROL_ATTACHED {
      type = "internal";
      number = <46>;
    };
    pin_define@NUM_CAMERAS {
      type = "internal";
      number = <1>;
    };
    pin_define@CAMERA_0_I2C_PORT {
      type = "internal";
      number = <0>;
    };
    pin_define@CAMERA_0_SDA_PIN {
      type = "internal";
      number = <28>;
    };
    pin_define@CAMERA_0_SCL_PIN {
      type = "internal";
      number = <29>;
    };
    pin_define@CAMERA_0_SHUTDOWN {
      type = "internal";
      number = <41>;
    };
    pin_define@CAMERA_0_UNICAM_PORT {
      type = "internal";
      number = <1>;
    };
    pin_define@CAMERA_0_LED {
      type = "internal";
      number = <32>;
    };
    pin_define@FLASH_0_ENABLE {
      type = "absent";
    };
    pin_define@FLASH_0_INDICATOR {
      type = "absent";
    };
    pin_define@FLASH_1_ENABLE {
      type = "absent";
    };
    pin_define@FLASH_1_INDICATOR {
      type = "absent";
    };
    pin_define@POWER_LOW {
      type = "internal";
      number = <35>;
    };
    pin_define@LEDS_DISK_ACTIVITY {
      type = "internal";
      number = <47>;
    };
    pin_define@LAN_RESET {
      type = "internal";
      number = <31>;
    };
  }; // pin_defines
```

```
            }; // pins
            pins_cm {
              pin_config {
                pin@default {
                  polarity = "active_high";
                  termination = "pull_down";
                  startup_state = "inactive";
                  function = "input";
                }; // pin
                pin@p14 { function = "uart0";  termination = "no_pulling";    }; // TX uart0
                pin@p15 { function = "uart0";  termination = "pull_up"; }; // RX uart0
                pin@p48 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD CLK
                pin@p49 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD CMD
                pin@p50 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D0
                pin@p51 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D1
                pin@p52 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D2
                pin@p53 { function = "sdcard"; termination = "pull_up";    drive_strength_mA = < 8 >; }; // SD D3
              }; // pin_config
              pin_defines {
              }; // pin_defines
            }; // pins_cm
  };
};
```