Programs Created

Python Codes

ArduinoSensors.Py

from nanpy import (ArduinoApi, SerialManager)

from config.SensorConfig import SensorConfig

class ArduinoSensors:

sensor\_config = None

arduino = None

HIGH\_STATE = None

LOW\_STATE = None

def \_\_init\_\_(self):

try:

serial\_manager = SerialManager()

self.arduino = ArduinoApi(connection = serial\_manager)

self.sensor\_config = SensorConfig()

self.arduino.pinMode(self.sensor\_config.getCap1(), self.arduino.INPUT)

self.arduino.pinMode(self.sensor\_config.getCap2(), self.arduino.INPUT)

self.arduino.pinMode(self.sensor\_config.getCap3(), self.arduino.INPUT)

self.arduino.pinMode(self.sensor\_config.getInd(), self.arduino.INPUT)

self.arduino.pinMode(self.sensor\_config.getLDRtb(), self.arduino.INPUT)

self.arduino.pinMode(self.sensor\_config.getLDRr(), self.arduino.INPUT)

self.arduino.pinMode(self.sensor\_config.getIR(), self.arduino.INPUT)

self.arduino.pinMode(self.sensor\_config.getSolN(),self.arduino.OUTPUT)

self.arduino.pinMode(self.sensor\_config.getSolD(),self.arduino.OUTPUT)

self.arduino.pinMode(self.sensor\_config.getSolR(),self.arduino.OUTPUT)

self.HIGH\_STATE = self.arduino.LOW

self.LOW\_STATE = self.arduino.HIGH

self.arduino.digitalWrite(self.sensor\_config.getSolN(), self.LOW\_STATE)

self.arduino.digitalWrite(self.sensor\_config.getSolD(), self.LOW\_STATE)

self.arduino.digitalWrite(self.sensor\_config.getSolR(), self.LOW\_STATE)

except:

from error.ErrorHandler import ErrorHandler

error\_handler = ErrorHandler()

error\_handler.handleError(location, message, args)

def getSensorState(self, sensor):

if sensor == 'ir':

self.arduino.pinMode(self.sensor\_config.getIR(), self.arduino.INPUT)

return self.arduino.digitalRead(self.sensor\_config.getIR())

elif sensor == 'cap1':

self.arduino.pinMode(self.sensor\_config.getCap1(), self.arduino.INPUT)

return self.arduino.digitalRead(self.sensor\_config.getCap1())

elif sensor == 'cap2':

self.arduino.pinMode(self.sensor\_config.getCap2(), self.arduino.INPUT)

return self.arduino.digitalRead(self.sensor\_config.getCap2())

elif sensor == 'cap3':

self.arduino.pinMode(self.sensor\_config.getCap3(), self.arduino.INPUT)

return self.arduino.digitalRead(self.sensor\_config.getCap3())

elif sensor == 'ind':

self.arduino.pinMode(self.sensor\_config.getInd(), self.arduino.INPUT)

return self.arduino.digitalRead(self.sensor\_config.getInd())

elif sensor == 'ldr\_tb':

self.arduino.pinMode(self.sensor\_config.getLDRtb(), self.arduino.INPUT)

return self.arduino.analogRead(self.sensor\_config.getLDRtb())

elif sensor == 'ldr\_r':

self.arduino.pinMode(self.sensor\_config.getLDRr(), self.arduino.INPUT)

return self.arduino.analogRead(self.sensor\_config.getLDRr())

def setSensorState(self, sensor, state):

if state == 'high':

state = self.HIGH\_STATE

elif state == 'low':

state = self.LOW\_STATE

if sensor == 'sol\_n':

self.arduino.digitalWrite(self.sensor\_config.getSolN(), state

elif sensor == 'sol\_d':

self.arduino.digitalWrite(self.sensor\_config.getSolD(), state)

elif sensor == 'sol\_r':

self.arduino.digitalWrite(self.sensor\_config.getSolR(), state)

This python source code is the program responsible to get the values of the Sensors of all types, inputted at the CRVDUINO. Notice the connection of the two microcontrollers via serial connection called in the codes' starting point. This allows the raspberry pi to determine what sensors activated upon sensing which is read by the CRVDUINO digital and analog pins.

Audio.py

import os

import pygame

class Audio:

script\_dir = None

sound\_path = None

def \_\_init\_\_(self):

pygame.mixer.init()

self.script\_dir = os.path.dirname(\_\_file\_\_)

self.sound\_path = os.path.join(self.script\_dir, '../assets/sounds/')

def sayWaste(self, classification, waste):

print self.sound\_path

print classification

print waste

play\_sound ="%s%s\_%s.mp3" % (self.sound\_path, waste, classification)

print play\_sound

pygame.mixer.music.load(play\_sound)

pygame.mixer.music.play()

while pygame.mixer.music.get\_busy() == True:

continue

This python source code executes the program for the speech recordings to play upon determining the classification of the input waste. The recordings will be played by the raspberry pi as the speaker could be directly connected to it. Speech recordings are on a specific folder where the path will be called by the play command.

GpioSensors.py

import RPi.GPIO as GPIO

from config.SensorConfig import SensorConfig

class GpioSensors:

sensor\_config = None

HIGH\_STATE = None

LOW\_STATE = None

def \_\_init\_\_(self):

try:

self.sensor\_config = SensorConfig()

GPIO.setmode(GPIO.BCM)

GPIO.setup(self.sensor\_config.getReadyLed(), GPIO.OUT)

GPIO.setup(self.sensor\_config.getScanningLed(), GPIO.OUT)

GPIO.setup(self.sensor\_config.getBiodegradableLed(), GPIO.OUT)

GPIO.setup(self.sensor\_config.getNonBiodegradableLed(), GPIO.OUT)

GPIO.setup(self.sensor\_config.getRecyclableLed(), GPIO.OUT)

GPIO.setup(self.sensor\_config.getOthersLed(), GPIO.OUT)

GPIO.setup(self.sensor\_config.getLDRLed(), GPIO.OUT)

GPIO.setup(self.sensor\_config.getIrGpio(), GPIO.IN)

self.HIGH\_STATE = GPIO.HIGH

self.LOW\_STATE = GPIO.LOW

except:

from error.ErrorHandler import ErrorHandler

error\_handler = ErrorHandler()

error\_handler.handleError(location, message, args)

self.cleanUpGpio()

def getSensorState(self, sensor):

if sensor == 'ir':

return GPIO.input(self.sensor\_config.getIrGpio())

def setSensorState(self, sensor, state):

if state == 'high':

state = self.HIGH\_STATE

elif state == 'low':

state = self.LOW\_STATE

if sensor == 'ready':

GPIO.output(self.sensor\_config.getReadyLed(), state)

elif sensor == 'scanning':

GPIO.output(self.sensor\_config.getScanningLed(), state)

elif sensor == 'bio':

GPIO.output(self.sensor\_config.getBiodegradableLed(), state)

elif sensor == 'non\_bio':

GPIO.output(self.sensor\_config.getNonBiodegradableLed(), state)

elif sensor == 'recyclable':

GPIO.output(self.sensor\_config.getRecyclableLed(), state)

elif sensor == 'others':

GPIO.output(self.sensor\_config.getOthersLed(), state)

elif sensor == 'ldr':

GPIO.output(self.sensor\_config.getLDRLed(), state)

def cleanUpGpio(self):

GPIO.cleanup()

As part of the raspberry pi microcontrollers, GPIO pins are utilized for some of the components; particularly the LED; IR and relays which signals are sent as HIGH or LOW in response to the classification of waste input. The LED embedded on the sensing box is being run by this program where there are different colors of LED for "Nagsisimula", "Naghihintay" at "Di-Matukoy" notification.

Sensors.py

from time import sleep

from components.ArduinoSensors import ArduinoSensors

from components.GpioSensors import GpioSensors

class Sensors:

arduino\_sensors = None

gpio\_sensors = None

def \_\_init\_\_(self):

self.arduino\_sensors = ArduinoSensors()

self.gpio\_sensors = GpioSensors()

def scan(self):

while True:

if self.gpio\_sensors.getSensorState('ir') == 0:

break

def getSensorValues(self):

cap1 = self.arduino\_sensors.getSensorState('cap1')

cap2 = self.arduino\_sensors.getSensorState('cap2')

cap3 = self.arduino\_sensors.getSensorState('cap3')

ind = self.arduino\_sensors.getSensorState('ind')

LDRtb = self.arduino\_sensors.getSensorState('ldr\_tb')

LDRr = self.arduino\_sensors.getSensorState('ldr\_r')

ir\_arduino = self.arduino\_sensors.getSensorState('ir')

sensor\_values = [int(cap1), int(cap2), int(cap3), int(ind), int(ir\_arduino), float(LDRtb), float(LDRr)];

self.testOuputScan(sensor\_values)

return sensor\_values

def testOuputScan(self, arr):

print "\n===================="

print "INPUTS"

print "cap1 => %i " % arr[0]

print "cap2 => %i " % arr[1]

print "cap3 => %i " % arr[2]

print "ind => %i " % arr[3]

print "Ir\_ard => %i" % arr[4]

print "LDRtb => %f " % arr[5]

print "LDRr => %i " % arr[6]

print "====================\n

def setArdionoSensorState(self, sensor, state):

print "%s %s" % (sensor, state)

self.arduino\_sensors.setSensorState(sensor, state)

def setGpioSensorState(self, sensor, state):

print "%s %s" % (sensor, state)

self.gpio\_sensors.setSensorState(sensor, state)

This source codes interface the output of all the sensors used for waste classification including the sensors which are on the GPIO and the ArduinoSensors. This program function to obtain the output of those sensors and use it for the database which is run by other programs.

View.py

import os

import psutil

from PIL import Image

class View:

script\_dir = None

image\_path = None

def \_\_init\_\_(self):

self.script\_dir = os.path.dirname(\_\_file\_\_)

self.image\_path = os.path.join(self.script\_dir, '../assets/images/')

def showText(self, text):

print self.image\_path

self.kill()

image = Image.open("%s%s.png" % (self.image\_path, text) )

image.show()

def showWaste(self, classification, waste):

self.kill()

image = Image.open("%s%s\_%s.png" % (self.image\_path, waste, classification) )

image.show()

def kill(self):

for proc in psutil.process\_iter():

if proc.name() == "display":

proc.kill()

def showError(self):

self.kill()

image = Image.open("%serror.png" % self.image\_path )

image.show()

#view = View()

#if 1:

# while str != 'exit':

# str = raw\_input('Enter the name of picture (exit to exit) -> ')

# view.showText(str)

This program is to get the image file stored on a specific folder in the raspberry pi. These images are outputted with respect to the input waste classification. Run by the show command, images will be displayed on the LCD upon calling with respect to the input classification.

MySqlConfig.py

class MySqlConfig:

URL = "localhost"

USERNAME = "root"

PASSWORD = "shabido"

DB\_NAME = "FinalProtoWC"

def getURL(self):

return self.URL

def getUsername(self):

return self.USERNAME

def getPassword(self):

return self.PASSWORD

def getDbName(self):

return self.DB\_NAME

MySql database is installed on the raspberry pi wherein configuration of the database is needed. This source code shows the configuration of the database created for the whole system including the username, password and the Database name which is FinalProtoWC.

SensorConfig.py

class SensorConfig:

#IR

IR\_PIN = 9

CAP1\_PIN = 13

CAP2\_PIN = 12

CAP3\_PIN = 11

IND\_PIN = 10

LDRtb\_PIN = 0

LDRr\_PIN = 1

SOL\_N = 7

SOL\_D = 6

SOL\_R = 5

#Gpio

IR\_GPIO\_PIN = 2

READY\_LED\_PIN = 3

SCANNING\_LED\_PIN = 4

BIODEGRADABLE\_LED\_PIN = 17

NON\_BIODEGRADABLE\_LED\_PIN = 27

RECYCLABLE\_LED\_PIN = 22

OTHERS\_LED\_PIN = 14

LDR\_LED\_PIN = 15

def getIR(self):

return self.IR\_PIN

def getCap1(self):

return self.CAP1\_PIN

def getCap2(self):

return self.CAP2\_PIN

def getCap3(self):

return self.CAP3\_PIN

def getInd(self):

return self.IND\_PIN

def getLDRtb(self):

return self.LDRtb\_PIN

def getLDRr(self):

return self.LDRr\_PIN

def getIrGpio(self):

return self.IR\_GPIO\_PIN

def getReadyLed(self):

return self.READY\_LED\_PIN

def getScanningLed(self):

return self.SCANNING\_LED\_PIN

def getBiodegradableLed(self):

return self.BIODEGRADABLE\_LED\_PIN

def getNonBiodegradableLed(self):

return self.NON\_BIODEGRADABLE\_LED\_PIN

def getOthersLed(self):

return self.OTHERS\_LED\_PIN

def getLDRLed(self):

return self.LDR\_LED\_PIN

def getRecyclableLed(self):

return self.RECYCLABLE\_LED\_PIN

def getOthersLed(self):

return self.OTHERS\_LED\_PIN

def getSolN(self):

return self.SOL\_N

def getSolD(self):

return self.SOL\_D

def getSolR(self):

return self.SOL\_R

This source codes shows the configuration of the sensors and components which are interfaced on both microcontrollers. This shows the pin assignment of the sensors and other components on the CRVDUINO board and Raspberry pi GPIO.

DataAccess.py

#This file is there to execute the db queries

from DB\_Connector import DB\_Connector

class DataAccess:

connection = None

def \_\_init\_\_(self):

db\_connector = DB\_Connector()

self.connection = db\_connector.getConnection()

def matchSensors(self, sensor\_values):

sql = """SELECT \* FROM tblSensor WHERE

cap1 = %i AND

cap2 = %i AND

cap3 = %i AND

ind = %i AND

ir = %i AND

LDRtb\_min <= %f AND

LDRtb\_max >= %f AND

LDRr\_min <= %f AND

LDRr\_max >= %f

""" % (int(sensor\_values[0]), int(sensor\_values[1]),

int(sensor\_values[2]), int(sensor\_values[3]), int(sensor\_values[4]),

float(sensor\_values[5]), float(sensor\_values[5]),

float(sensor\_values[6]), float(sensor\_values[6]))

try:

cursor = self.connection.cursor()

cursor.execute(sql)

#get waste input

last\_id = self.getLastId('tblForcedataGeneration') #CHECK IF TO CHAR

last\_id = last\_id + 1

waste\_input = "waste%i" % last\_id

results = cursor.fetchall()

if cursor.rowcount == 0:

return ['unknown', 'unknown', waste\_input, sensor\_values[0], sensor\_values[1], sensor\_values[2], sensor\_values[3], sensor\_values[4], sensor\_values[5], sensor\_values[6], 0]

elif cursor.rowcount > 1:

from error.ErrorHandler import ErrorHandler

error\_handler = ErrorHandler()

error\_handler.handleError(location, message, args)

self.closeDB()

else:

return [results[0][1], results[0][2], waste\_input, results[0][3], results[0][4], results[0][5], results[0][6], results[0][7], sensor\_values[5], sensor\_values[6], 1]

except:

from error.ErrorHandler import ErrorHandler

error\_handler = ErrorHandler()

error\_handler.handleError(location, message, args)

self.closeDB()

def record(self, args):

sql = """ INSERT INTO tblForcedataGeneration

(waste\_input, cap1, cap2, cap3, ind, ir, LDRtb, LDRr, classified\_status)

VALUES ('%s', %i, %i, %i, %i, %i, %f, %f, %i)

""" % (str(args[0]), int(args[1]), int(args[2]), int(args[3]), int(args[4]), int(args[5]), float(args[6]), float(args[7]), int(args[8]))

try:

cursor = self.connection.cursor()

cursor.execute(sql)

self.connection.commit()

except:

from error.ErrorHandler import ErrorHandler

error\_handler = ErrorHandler()

error\_handler.handleError(location, message, args)

self.closeDB()

def tally(self, classification, material):

sql = """

SELECT frequency\_count FROM tblOutputReport

WHERE classification = '%s' AND

material = '%s'

""" % (classification, material)

#try:

cursor = self.connection.cursor()

cursor.execute(sql)

frequency\_count = cursor.fetchone()

new\_frequency\_count = frequency\_count[0] + 1

sql = """

UPDATE tblOutputReport

SET frequency\_count = %i WHERE

classification = '%s' AND

material = '%s'

""" % (new\_frequency\_count, classification, material)

cursor.execute(sql)

self.connection.commit()

#except:

#from error.ErrorHandler import ErrorHandler

#error\_handler = ErrorHandler()

#error\_handler.handleError(location, message, args)

#self.closeDB()

def getLastId(self, table):

sql = """SELECT id

FROM %s

ORDER BY id DESC

LIMIT 1

""" % (table)

cursor = self.connection.cursor()

cursor.execute(sql)

last\_id = cursor.fetchone()

return last\_id[0]

def closeDB(self):

self.connection.close()

This source code makes the program to run the queries done by the database. Tables from the database are generated through this program, where *tblForcedataGeneration, tblSensor, and tblOutputReport* are generated with respect to the inputs on the system.

DB\_Connector.py

import MySQLdb

from config.MySqlConfig import MySqlConfig

class DB\_Connector:

connection = None

#establish a connection

def \_\_init\_\_(self):

db\_config = MySqlConfig()

try:

self.connection = MySQLdb.connect(db\_config.getURL(), db\_config.getUsername(), db\_config.getPassword(), db\_config.getDbName())

except:

from error.ErrorHandler import ErrorHandler

error\_handler = ErrorHandler()

error\_handler.handleError(location, message, args)

def getConnection(self):

return self.connection

The source code name implies that it is the program needed for the connection of the database to the python language wherein terminal commands would execute it upon calling.

SystemImplementor.py

from components.Audio import Audio

from components.Sensors import Sensors

from components.View import View

from database.DataAccess import DataAccess

import time

class SystemImplementor:

audio = None

sensors = None

view = None

data\_access = None

def \_\_init\_\_(self):

self.audio = Audio()

self.sensors = Sensors()

self.view = View()

self.data\_access = DataAccess()

def start(self):

self.ready()

self.sensors.scan()

self.scanning()

sensor\_values = self.sensors.getSensorValues()

time.sleep(1)

results = self.data\_access.matchSensors(sensor\_values)

#classification = results[0]

#classification = 'nareresiklo'

classification = 'dinabubulok'

#classification = 'nabubulok'

#waste = results[1]

waste = 'plastic'

self.results(classification, waste)

record\_values = [results[2], results[3], results[4], results[5], results[6], results[7], results[8], results[9], results[10] ]

self.iterateDatabase(classification, waste, record\_values)

return True

def iterateDatabase(self, classification, material, record\_values):

self.data\_access.record(record\_values)

self.data\_access.tally(classification, material)

def ready(self):

self.view.showText('init')

self.sensors.setGpioSensorState('ready', 'high')

time.sleep(.3)

self.sensors.setGpioSensorState('ready', 'low')

time.sleep(.7)

self.sensors.setGpioSensorState('ready', 'high')

time.sleep(.3)

self.sensors.setGpioSensorState('ready', 'low')

time.sleep(.7)

self.sensors.setGpioSensorState('ready', 'high')

time.sleep(.3)

self.sensors.setGpioSensorState('ready', 'low')

self.view.showText('ready')

def scanning(self):

self.view.showText('scanning')

self.sensors.setGpioSensorState('scanning', 'high')

self.sensors.setGpioSensorState('ldr', 'high')

time.sleep(2)

def results(self, classification, waste):

self.view.showWaste(classification, waste)

time.sleep(.3)

self.sensors.setGpioSensorState('scanning', 'low')

self.sensors.setGpioSensorState('ldr', 'low')

if(classification == 'nareresiklo'):

self.sensors.setGpioSensorState('recyclable', 'high')

self.sensors.setArdionoSensorState('sol\_r', 'high')

self.audio.sayWaste(classification, waste)

time.sleep(3)

self.sensors.setGpioSensorState('recyclable', 'low')

self.sensors.setArdionoSensorState('sol\_r', 'low')

if(classification == 'dinabubulok'):

self.sensors.setGpioSensorState('non\_bio', 'high')

self.sensors.setArdionoSensorState('sol\_d', 'high')

self.audio.sayWaste(classification, waste)

time.sleep(3)

self.sensors.setGpioSensorState('non\_bio', 'low')

self.sensors.setArdionoSensorState('sol\_d', 'low')

if(classification == 'nabubulok'):

self.sensors.setGpioSensorState('bio', 'high')

self.sensors.setArdionoSensorState('sol\_n', 'high')

self.audio.sayWaste(classification, waste)

time.sleep(3)

self.sensors.setGpioSensorState('bio', 'low')

self.sensors.setArdionoSensorState('sol\_n', 'low')

if(classification == 'unknown'):

self.sensors.setGpioSensorState('others', 'high')

self.audio.sayWaste(classification, waste)

time.sleep(3)

self.sensors.setGpioSensorState('others', 'low')

This program is the one responsible for the coordination of the other python codes, to which programs are implemented to run as a whole system.

C++ Codes

ArduinoClass.cpp

#include <Arduino.h>

#include "ArduinoClass.h"

#include <stdlib.h>

const char\* nanpy::ArduinoClass::get\_firmware\_id()

{

return "A";

}

void nanpy::ArduinoClass::elaborate( nanpy::MethodDescriptor\* m ) {

if (strcmp(m->getName(), "dw") == 0) { // digitalWrite

digitalWrite(m->getInt(0), m->getInt(1));

m->returns(0);

}

if (strcmp(m->getName(), "r") == 0) { // digitalRead

m->returns(digitalRead(m->getInt(0)));

}

if (strcmp(m->getName(), "aw") == 0) { // analogWrite

analogWrite(m->getInt(0), m->getInt(1));

m->returns(0);

}

if (strcmp(m->getName(), "a") == 0) { // analogRead

m->returns(analogRead(m->getInt(0)));

}

if (strcmp(m->getName(), "pm") == 0) { // pinMode

pinMode(m->getInt(0), m->getInt(0);

m->returns(0);

}

if (strcmp(m->getName(), "delay") == 0) {

m->returns(0);

}

if (strcmp(m->getName(), "m") == 0) { // millis

m->returns(millis());

}

if (strcmp(m->getName(), "pi") == 0) { // pulseIn

pulseIn(m->getInt(0), m->getInt(0));

m->returns(0);

}

if (strcmp(m->getName(), "s") == 0) { // shiftOut

// shiftOut(dataPin, clockPin, bitOrder, value)

shiftOut(m->getInt(0), m->getInt(1), m->getInt(2), m->getInt(3));

m->returns(0);

}

};

ArduinoCoreClass.cpp

#include "cfg\_all.h"

#if USE\_ArduinoCore

#include <Arduino.h>

#include "ArduinoCoreClass.h"

#include <stdlib.h>

#define RETURN\_PIN\_FUNC(x) if (strcmp(m->getName(), #x) == 0) { m->returns(x(m->getInt(0))); }

#define RETURN\_PORT\_FUNC(x) if (strcmp(m->getName(), #x) == 0) { m->returns((uint32\_t)x(m->getInt(0))); }

const char\* nanpy::ArduinoCoreClass::get\_firmware\_id()

{

return "Core";

}

void nanpy::ArduinoCoreClass::elaborate(nanpy::MethodDescriptor\* m)

{

RETURN\_PIN\_FUNC(digitalPinToBitMask);

RETURN\_PIN\_FUNC(digitalPinToPort);

RETURN\_PIN\_FUNC(digitalPinToTimer);

if (strcmp(m->getName(), "analogInPinToBit") == 0)

{

#ifdef analogInPinToBit

m->returns(analogInPinToBit(m->getInt(0)));

#else

m->returns(-1);

#endif

}

// RETURN\_PIN\_FUNC(analogInPinToBit);

// if (strcmp(m->getName(), "portModeRegister") == 0)

// {

// m->returns((uint32\_t)portModeRegister(m->getInt(0)));

// }

RETURN\_PORT\_FUNC(portModeRegister);

RETURN\_PORT\_FUNC(portOutputRegister);

RETURN\_PORT\_FUNC(portInputRegister);

}

#endif

These two C++ programs are used to make the CRVDUINO as the slave to the Raspberry Pi microcontroller. This implies the ability of the Raspberry Pi to control the CRVDUINO via python codes.