OSU Mechanical Engineering

Smart Products Laboratory

Computer Vision | Nine

ME Course Number
Spring 2019

Table of Contents

1 Overview

In this lab you will be developing a custom communication protocol over an asynchronous serial line. You will also be developing algorithms to detect different colored circles and reporting them from the camera to the RPi.

2 Background

See references

3 References & Resources

3.1 References

The following references may be helpful to complete the lab.

1. https://www.elastic.io/message-exchange-patterns-application-integration/
2. http://garba.org/article/general/soa/mep.html
3. http://docs.openmv.io/library/pyb.UART.html
4. http://docs.openmv.io/library/omv.sensor.html
5. http://docs.openmv.io/library/omv.image.html#class-statistics-statistics-object
6. http://docs.openmv.io/library/omv.image.html#class-histogram-histogram-object
7. http://docs.openmv.io/library/omv.image.html#class-circle-circle-object

4 Pre-lab

None

5 Laboratory

Save all code, data, and solutions as you will need them to complete the final lab report.

5.1 Requirements

Bring your raspberry pi (RPI) so that you can develop your code and test your GPIO class.

5.2 Procedure

The serial communication lines consist of a transmit line (TX) and a receive line (RX). You will use a 4 wire USB-TTL cable to communicate with the camera from the RPi. The RPi will act as the master and the camera as the slave. You need to plus in the USB end of the cable into the RPi and attach the TX (green wire to camera RX) and RX (white wire to camera TX) lines to their complementary pins on the camera. The following example scripts demonstrate how to program the RPi and the camera to communicate with one another. Run these scripts after setting up your hardware to assure you've correctly assembled the communication lines.

```
#Example UART communication between RPi and OpenMV By Dylan DeSantis
##################################
# This program demonstrates how to communicate over uart
# with the raspberry pi.
##################################

import sensor, image, time
from pyb import UART
#Sensor Setup
##################################
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
##################################

#Communication Setup
##################################
uart = UART(1, 9600, timeout_char=1000)                         # init with
given baudrate
uart.init(9600, bits=8, parity=None, stop=1, timeout_char=1000) # init with
given parameters
##################################

#Main Loop
##################################
clock = time.clock()
while(True):
    clock.tick()
    while(uart.any() ==0):
        continue
    print(20)
    while(uart.any() !=0):
        print(uart.readchar())
    time.sleep(1000)
    uart.writechar(8)
##################################
```

```
/*************************************************************
                  Comms_Example.cpp
*************************************************************
Simple program to communicate with the openMV cam using the
USB  port. Use the  USB-TLL cables for the Pi and connect the
green (tx) and white(rx) leads to the RX(P0) and and TX(P1) pins
on the OpenMV camera, respectively.

To compile:
sudo g++ -o comms Comms_Example.cpp -lwiringPi
*************************************************************
    @author     Dylan DeSantis
    @date       3/25/2018
    @version    1.0.0
*************************************************************/
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
#include<errno.h>
#include<math.h>
#include<signal.h>
#include<unistd.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#include<sstream>
#include<fstream>
#include<string.h>

int main (void)
{
    int fd1 = serialOpen ("/dev/ttyUSB0", 9600);
    wiringPiSetup ();
    fflush (stdout);
    int i = 0;
    int data = 0;
    while(i<10)
    {
            int send_data = 5;
            serialPutchar (fd1, send_data);
            int num = 0;
            while(num==0)
            {
                num =serialDataAvail(fd1);
                usleep(1000);
            }
            data = serialGetchar(fd1);
            printf("Data Recieved: %d \n", data);
            i++;
        }
      serialClose(fd1);
  return 0;
}
```

Once you've verified your setup is working, you will now proceed to develop the camera code. Firstly, you will need to design an algorithm to identify circles. The API on the openMV platform provides a built-in function named find_circles. You will have to adjust the parameters of the function in order to enable the best performance for detecting the ping pong balls. After doing this you will need to identify the color of the ping pong balls by acquiring the RGB information of the region where the circles were previously identified. You will need to convert the color information from CSV to RGB. See the functions below.

```
1. Img = sensor.snapshot()
2. list_of_circles = Img.find_circles(…)
3. hist = Img.get_histogram(…)
4. stat = hist.get_statistics()
5. rgb = image.lab_to_rgb(…)
6. Img.draw.circle(…)
```

The two main pieces of information you must report from the camera to the RPi are,

1. The pixel location (X, Y in the image coordinate system) of the center of the circle.
2. The color of the circle, which can be in the form of either the RGB value of the circle region or an indication of whether the circle region is red, blue, or green.

Other pieces of information you may want to include is the radius of the detected circles and the number of detected circles. Together we can think of this as a function called on the **get_ball_info**.

You will also need to develop the necessary software to be able to communicate this information. You may use any logic you wish to transmit the information to the RPi over the TX and RX lines. However, the following structure is provided to assist in developing your protocol.

The communication protocol of the lines between the RPi and the camera should consist of a request from the RPi to deliver a command to the camera. This transmission will have a timeout period in which the RPi waits a certain amount of time for the camera to send an acknowledgement packet, informing the RPi that it may send its command to the camera. Once the RPi sends an appropriate command, the camera will launch a corresponding function. The function can send back data or just run a function that returns no data, in which case the camera would send a packet upon its completion informing the RPi that it can proceed to request another command. From the perspective of the RPi the packets would look like:

RPi: RQST::------::CMD1::--------::

CAM: -------::ACK::-------::DATA::

You can make these packets any character you want for example: RQST = 0x01, CMD1 = 0x02, ACK = 0x03. You just need to make sure that the programs for both the RPi and the camera are in agreement of the meaning of these values.

On the camera platform

1. The main function should be an infinite loop called wait_for_command()

2. Inside this function you should
    a. Check to see if a RQST has been administered by the RPi. If so, send an ACK packet and wait for a CMD. Otherwise, continue waiting for a RQST packet.
    b. Once a CMD has been transmitted you should check to see which function corresponds to the CMD. If there is no corresponding function, then an error packet should be transmitted to the RPi. Otherwise, the function should be called.
        i. Inside the function you should have the proper routine set so that the camera performs the needed action. The camera can then send information produced from the routine or simply send an END packet.
    c. At the end of this function you should then exit the wait_for_command function which would then end the loop and the camera would begin waiting for another RQST packet.

    One of these functions might look like this.

    1. Run get_ball_info()
        a. Find_circles(…)
        b. Do some logic to select only circles that probably correspond to ping pong balls.
        c. Count the number of balls in the image.
        d. Send the number of balls found in the frame.
        e. Wait for an acknowledgement from the RPi.
        f. For n balls send the packet INFO
            i. INFO: BEGIN::X_location :: Y_location ::Color ::Radius::END
        g. Exit routine

Once you finish your software for both the RPi and the camera software, show the GTA your working implementation.