# SONBI ROBOT HUMAN DETECTION USING KINECT AND RASPBERRY PI

## Project Report
## Course: Intelligent Robotics II
## Term: Winter 2014

**Name: Wajid Ali Mohammed Ali**
**PSU ID: 928655091**

# INDEX

## A. OBJECTIVE

To build the software system of the raspberry pi inside Sonbi and integrate the Microsoft Kinect onto to Raspberry pi and make it interactive with Sonbi robot in a way when people stands in front of Kinect, the Sonbi robot waves his arms to the people.

## B. HARDWARE SYSTEM

Sonbi has the following basic hardware items mounted inside of his chest chassis:

- Raspberry Pi w/8GB Flash
- Pololu Maestro 24
- Microsoft Kinect
- ATX 500 Watt PS
- Misc parts (proto boards, wire, mechanical)

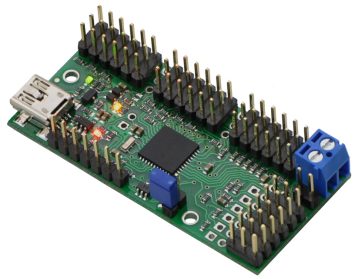
### Raspberry pi:

- 700 MHz ARM1176JZF-S core processor
- 512 MB SDRAM
- Powered via microUSB (5V)
- Ethernet, HDMI, and 2 USB ports for peripherals
- Raspbian OS
- Widely used, lots of documentation!

### Pololu maestro 24

- 24 Channels
- Pulse rate up to 333Hz
- Script size up to 8KB

- Up to 1.5 amps per channel
- 2 Power Options USB/power header
- Scripting or native API support



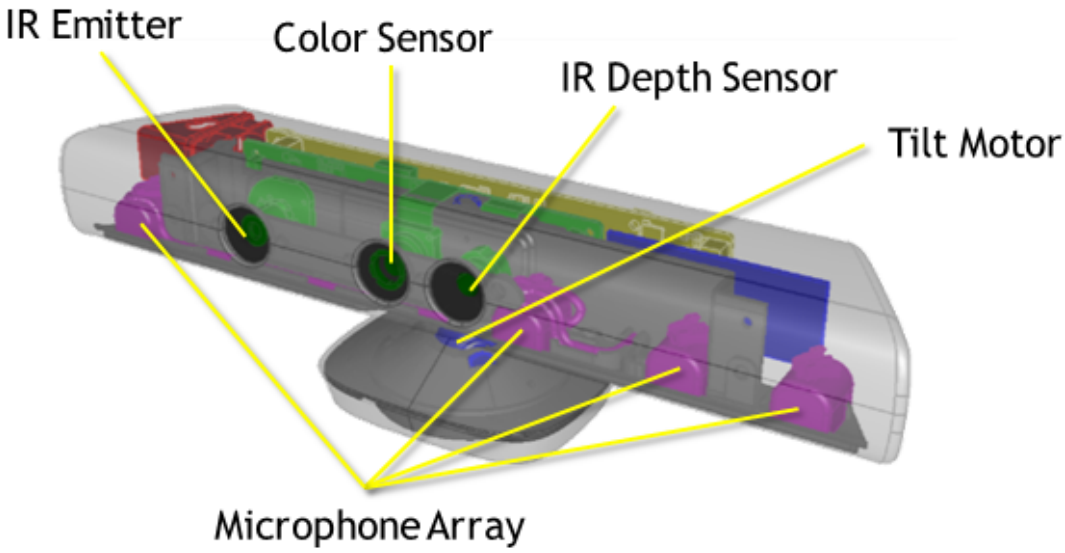
**Raspberry pi and Pololu interface**

- Simple wiring
  - Power, Gnd, Tx-Rx, and Rx-Tx
- TTL serial port
  - By default Pi uses serial port for console input/output
  - Edit /etc/inittab and /boot/cmdline.txt to change default and free serial port for use
- Great tutorial at: http://shahmirj.com/blog/raspberry-pi-and-pololu-servo-controller-using-c

**Microsoft Kinect:**

- An RGB camera that stores three channel data in a 1280x960 resolution. This makes capturing a color image possible.
- An infrared (IR) emitter and an IR depth sensor. The emitter emits infrared light beams and the depth sensor reads the IR beams reflected back to the sensor. The reflected beams are converted into depth information measuring the distance between an object and the sensor. This makes capturing a depth image possible.
- A multi-array microphone, which contains four microphones for capturing sound. Because there are four microphones, it is possible to record audio as well as find the location of the sound source and the direction of the audio wave.
- A 3-axis accelerometer configured for a 2G range, where G is the acceleration due to gravity. It is possible to use the accelerometer to determine the current orientation of the Kinect

- Vertical Tilt angle: 27 deg
- Frame Rate :30 fps


**C.INTEGRATING THE KINECT WITH RASPBERRY PI**

The following are the steps taken to integrate the Kinect with Raspberry Pi.

**Connecting the Microsoft Kinect and its sensor drivers on the raspberry pi:**

This process is one of the tedious parts of the project, as one must be aware that Kinect works on windows and to make it work on unix based operating system we need to manually install all the libraries

and drivers associated with it which is hard and takes a lot of man hours in resolving the issues. The steps taken and libraries and list of packages installed are given in the section building software systems.

**Using Kinect's full capability:**

To use all the features of Kinect such as depth sensors, IR sensor, mic and motors to tilt the camera we need libraries that can do this. The RPI by default has OpenCV and Open GL/GLES mounted on it, but these doesn't support (very soon it will) depth sensors and motors yet, so we need OpenNI or Libfreenect package to be installed. Either one is enough but I decided to install both. To test and understand, you can run sample programs, which are available in the OpenNI and Libfreenect folders. I have already complied and built the binaries. One can run it just go the "bin" folder and running the samples by ./"sample program."

## D. BUILDING SOFTWARE SYSTEM

**Libfreenect:**

Libfreenect is a userspace driver for the Microsoft Kinect. It runs on Linux supports

- RGB and Depth Images
- Motors
- Accelerometer
- LED

Audio is a work in progress

To build libfreenect, you'll need

- libusb >= 1.0.13
- CMake >= 2.6
- python == 2.* (only if BUILD_AUDIO or BUILD_PYTHON)

For the examples, you'll need

- OpenGL (included with OSX)
- glut (included with OSX)
- pthreads-win32 (Windows)

```
git clone https://github.com/OpenKinect/libfreenect cd libfreenect mkdir build cd build cmake -L .. make  # if you don't have `make` or don't want color output # cmake --build
```

```
sudo apt-get install git-core cmake pkg-config build-essential libusb-1.0-0-dev
sudo adduser $USER video sudo adduser $USER plugdev # necessary?  # only if you are
building the examples: sudo apt-get install libglut3-dev libxmu-dev libxi-dev
```

**Wrappers:**

Interfaces to various languages are provided in wrappers/. Wrappers are not guaranteed to be API stable or up to date.

- C (using a synchronous API)
- C++
- C#
- python
- ruby
- actionscript
- Java (JNA)

**OpenNI:**

Requirements:

1) GCC 4.x
From: http://gcc.gnu.org/releases.html
Or via apt: sudo apt-get install g++

2) Python 2.6+/3.x
From: http://www.python.org/download/
Or via apt:  sudo apt-get install python

3) LibUSB 1.0.x
From: http://sourceforge.net/projects/libusb/files/libusb-1.0/
Or via apt: sudo apt-get install libusb-1.0-0-dev

4) FreeGLUT3
From: http://freeglut.sourceforge.net/index.php#download
Or via apt:  sudo apt-get install freeglut3-dev

5) JDK 6.0
From:
http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u32-downloads-1594644.html
Or via apt:  sudo add-apt-repository "deb

Optional Requirements (To build the documentation):

1) Doxygen
From:
http://www.stack.nl/~dimitri/doxygen/download.html#latestsrc
Or via apt: sudo apt-get install doxygen

2) GraphViz
From: http://www.graphviz.org/Download_linux_ubuntu.php
Or via apt: sudo apt-get install graphviz

**Building OpenNI:**

1) Go into the directory: "Platform/Linux/CreateRedist".
Run the script: "./RedistMaker".
This will compile everything and create a redist package in the "Platform/Linux/Redist" directory. It will also create a distribution in the "Platform/Linux/CreateRedist/Final" directory.

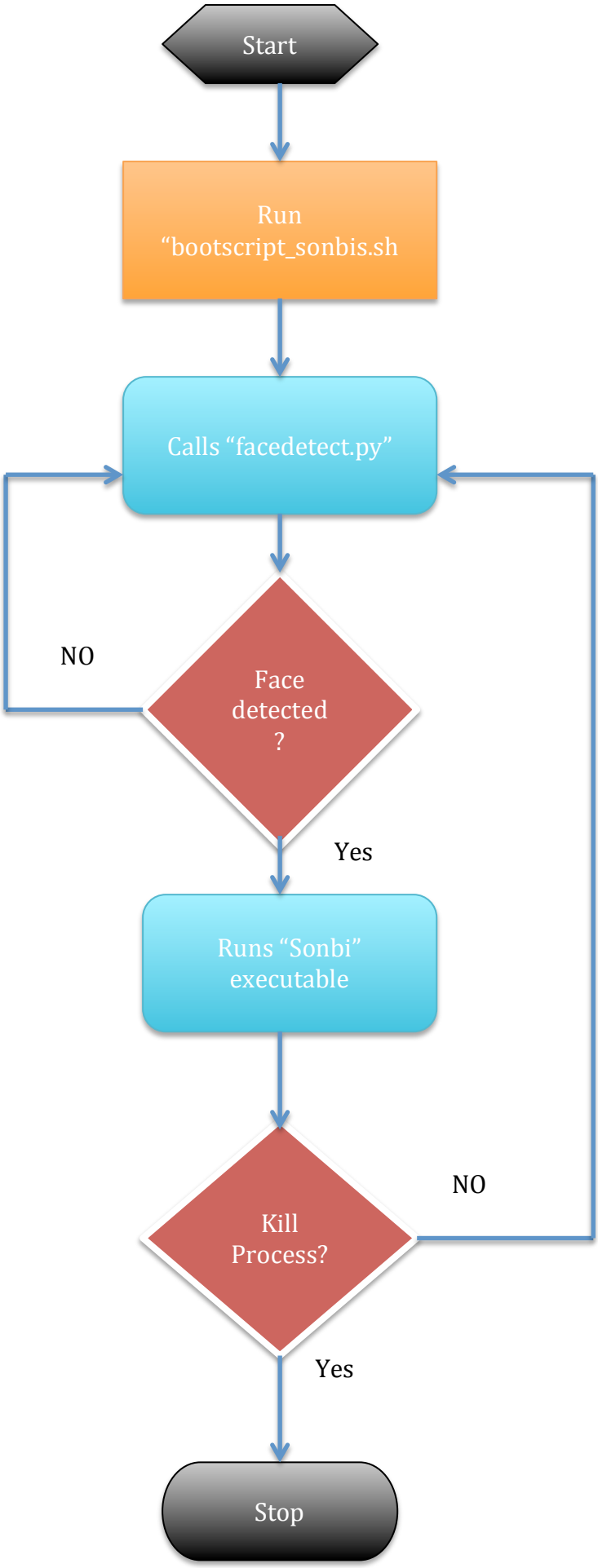
2) Go into the directory: "Platform/Linux/Redist".
Run the script: "sudo ./install.sh" (needs to run as root)
The install script copies key files to the following location:
Libs into: /usr/lib
Bins into: /usr/bin
Includes into: /usr/include/ni
Config files into: /var/lib/ni
If you wish to build the Mono wrappers, also run "make mono_wrapper" and "make mono_samples"

**E. PERSON DETECTION AND SONBI'S ACTION:**

The Raspberry PI runs a program **bootscript_sonbi.sh**

The "**bootscript_sonbi.sh**" run the command "**python facedetect.py --cascade=face.xml 0"**
You will need to download this trained face file:
http://stevenhickson-code.googlecode.com/svn/trunk/AUI/Imaging/face.xml

The **facedetect.py** runs the face detection algorithm and triggers the "Sonbi" executable.The Sonbi binary is responsible for bringing the servo motors into action. The flowchart of the process is below.


Start
Run
“bootscript_sonbis.sh
Calls “facedetect.py”
Face
detected
?
NO
Yes
Runs “Sonbi”
executable
Kill
Process?
NO
Yes
Stop

**PROJECT CODE:**

## Facedetect.py:

```
#!/usr/bin/python

The program finds faces in a camera image or video stream and
displays a red box around them.

import sys
import cv2.cv as cv
from optparse import OptionParser

min_size = (20, 20)
image_scale = 2
haar_scale = 1.2
min_neighbors = 2
haar_flags = 0

def detect_and_draw(img, cascade):
# allocate temporary images
gray = cv.CreateImage((img.width,img.height), 8, 1)
small_img = cv.CreateImage((cv.Round(img.width / image_scale),
            cv.Round (img.height / image_scale)), 8, 1)

# convert color input image to grayscale
cv.CvtColor(img, gray, cv.CV_BGR2GRAY)

# scale input image for faster processing
cv.Resize(gray, small_img, cv.CV_INTER_LINEAR)
cv.EqualizeHist(small_img, small_img)

if(cascade):
t = cv.GetTickCount()
faces = cv.HaarDetectObjects(small_img, cascade,
cv.CreateMemStorage(0), haar_scale, min_neighbors, haar_flags,
min_size)

t = cv.GetTickCount() - t
print "time taken for detection = %gms" %
(t/(cv.GetTickFrequency()*1000.))

if faces:

os.system(Sonbi) //Calling Sonbi binary to the do the action

for ((x, y, w, h), n) in faces:
# the input to cv.HaarDetectObjects was resized, so scale the
# bounding box of each face and convert it to two CvPoints
pt1 = (int(x * image_scale), int(y * image_scale))
pt2 = (int((x + w) * image_scale), int((y + h) * image_scale))
cv.Rectangle(img, pt1, pt2, cv.RGB(255, 0, 0), 3, 8, 0)
```

```
cv.ShowImage("video", img)

if __name__ == '__main__':

parser = OptionParser(usage = "usage: %prog [options]
[filename|camera_index]")
parser.add_option("-c", "--cascade", action="store",
dest="cascade", type="str", help="Haar cascade file, default
%default", default =
"../data/haarcascades/haarcascade_frontalface_alt.xml")
(options, args) = parser.parse_args()

cascade = cv.Load(options.cascade)

if len(args) != 1:
parser.print_help()
sys.exit(1)

input_name = args[0]
if input_name.isdigit():
capture = cv.CreateCameraCapture(int(input_name))
else:
capture = None

cv.NamedWindow("video", 1)

#size of the video
width = 160
height = 120

if width is None:
width = int(cv.GetCaptureProperty(capture,
cv.CV_CAP_PROP_FRAME_WIDTH))
else:
cv.SetCaptureProperty(capture,cv.CV_CAP_PROP_FRAME_WIDTH,width
)

if height is None:
height = int(cv.GetCaptureProperty(capture,
cv.CV_CAP_PROP_FRAME_HEIGHT))
else:
cv.SetCaptureProperty(capture,cv.CV_CAP_PROP_FRAME_HEIGHT,heig
ht)

if capture:
frame_copy = None
while True:

frame = cv.QueryFrame(capture)
if not frame:
cv.WaitKey(0)
break
if not frame_copy:
frame_copy = cv.CreateImage((frame.width,frame.height),
```

```
                                cv.IPL_DEPTH_8U, frame.nChannels)

if frame.origin == cv.IPL_ORIGIN_TL:
cv.Copy(frame, frame_copy)
else:
cv.Flip(frame, frame_copy, 0)

detect_and_draw(frame_copy, cascade)

if cv.WaitKey(10) >= 0:
break
else:
image = cv.LoadImage(input_name, 1)
detect_and_draw(image, cascade)
cv.WaitKey(0)

cv.DestroyWindow("video")
```

**Pololu.CPP**

```
/**
* This is a library of functions that send commands to a
Pololu
* Mini Maestro servo controller.
*
*
* This program makes use of sample code by Shahmir Javaid from
the
* tutorial Raspberry Pi and Pololu Servo Controller using C,
which is
* available at shahmirj.com/
*
* @author Stephanie Wallick
*     Winter 2014
*/

#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <termios.h>
#include <math.h>

#include "pololu.h"

/**
 * Sets targets for all channels on servo controller. Channels
are zero-indexed. Function
 * expects an array of target values where the channel number
corresponds to the index value.
 *
 * Use this function when you want to move multiple servos
simultaneously.
 *
 * Returns 0 for success or -1 if command failed to write to
```

```
controller.
 */
int maestroSetAllTargets(int fd, unsigned short
target_array[])
{
    unsigned char first = POLOLU_FIRST_CHANNEL;
    unsigned char size = POLOLU_LAST_CHANNEL + 1;

    unsigned char command[] = {0xAA, 0xC, 0x1F, size, first,

target_array[0] & 0x7F, target_array[0]  >> 7 & 0x7F,
target_array[1] & 0x7F, target_array[1]  >> 7 & 0x7F,
target_array[2] & 0x7F, target_array[2]  >> 7 & 0x7F
target_array[3] & 0x7F, target_array[3]  >> 7 & 0x7F,
target_array[4] & 0x7F, target_array[4]  >> 7 & 0x7F,
target_array[5] & 0x7F, target_array[5]  >> 7 & 0x7F,
target_array[6] & 0x7F, target_array[6]  >> 7 & 0x7F,
target_array[7] & 0x7F, target_array[7]  >> 7 & 0x7F,
target_array[8] & 0x7F, target_array[8]  >> 7 & 0x7F,
target_array[9] & 0x7F, target_array[9]  >> 7 & 0x7F,
target_array[10]& 0x7F, target_array[10] >> 7 & 0x7F,
target_array[11]& 0x7F, target_array[11] >> 7 & 0x7F,
target_array[12]& 0x7F, target_array[12] >> 7 & 0x7F,
target_array[13]& 0x7F, target_array[13] >> 7 & 0x7F,
target_array[14]& 0x7F, target_array[14] >> 7 & 0x7F,
target_array[15]& 0x7F, target_array[15] >> 7 & 0x7F,
target_array[16]& 0x7F, target_array[16] >> 7 & 0x7F,
target_array[17]& 0x7F, target_array[17] >> 7 & 0x7F,
target_array[18]& 0x7F, target_array[18] >> 7 & 0x7F,
target_array[19]& 0x7F, target_array[19] >> 7 & 0x7F,
target_array[20]& 0x7F, target_array[20] >> 7 & 0x7F,
target_array[21]& 0x7F, target_array[21] >> 7 & 0x7F,
target_array[22]& 0x7F, target_array[22] >> 7 & 0x7F,
target_array[23]& 0x7F, target_array[23] >> 7 & 0x7F};

    if (write(fd, command, sizeof(command)) == -1) {
        perror("error writing");
    return -1;
    }

    return 0;

}

/**
* Determines whether the servo outputs have reached their
targets or are still changing.
*
* Returns 0 if no servos are moving or 1 if servos are moving.
*/

int maestroGetMovingState(int fd)
{
    unsigned char command[] = {0xAA, 0xC, 0x13};
```

```
    if (write(fd, command, sizeof(command)) != 3)
    {
        perror("error writing");
        return -1;
    }

    unsigned char response[1];

    int ec = read(fd, response, 1);

    if(ec < 0) {
        perror("error reading");
        return ec;
    }

    return response[0];
}

/**
* Sets the acceleration limit of a servo channel's output.
Acceleration value can
* range from 0 to 255. Acceleration is in units of 0.25us /
10ms / 80ms.
*
* Note: An acceleration value of 0 correspondes to no
acceleration limit.
*
* returns 0 if read is successful or negative error if read
fails.
*/

int maestroSetAcceleration( int fd, unsigned char channel,
unsigned short acceleration)
{
    unsigned char command[] = {0xAA, 0xC, 0x09, channel,
acceleration & 0x7F, acceleration >> 7 & 0x7F};
    if (write(fd, command, sizeof(command)) == -1)
    {
        perror("error writing");
        return -1;
    }

    return 0;

}

/**
* Sets the limit of the speed at which a servo channel's
output value changes.
* The speed limit is given in units of 0.25us / 10ms. For
example, passing this
* function a speed value of 140 corresponds to a speed of
3.5us/ms (140 * 0.25/10)
* which means that it will take 100 ms to adjust the target
```

```
from 1000 to 1350 us.
*
* Note: setting the speed to 0 makes the speed unlimited (i.e.
a target will
* immediately affect position).
*
* returns 0 if read is successful or negative error if read
fails.
*/

int maestroSetSpeed(int fd, unsigned char channel, unsigned
short speed)
{
    unsigned char command[] = {0xAA, 0xC, 0x07, channel, speed
& 0x7F, speed >> 7 & 0x7F};
    if (write(fd, command, sizeof(command)) == -1)
    {
        perror("error writing");
        return -1;
    }

    return 0;
}

/**
* This command will return the error number on the Pololu
Maestro Board
* The error number is made of two bytes. so the response needs
to add the
* returned bytes together. Note that reading the error
register on the
* Pololu clears the error bits.
*
* @param int fd - The file descriptor to the device
*
* returns contents of error register if read is successful or
negative  error
* if read fails.
*/

int maestroGetError(int fd)
{
    unsigned char command[] = { 0xAA, 0xC, 0x21 };
    if (write(fd, command, sizeof(command)) != 3)
    {
        perror("error writing");
        return -1;
    }

    int n = 0;
    unsigned char response[2];
    do
    {
        int ec = read(fd, response+n, 1);
```

```
        if(ec < 0)
        {
            perror("error reading");
            return ec;
        }
        if (ec == 0)
        {
            continue;
        }
        n++;

    } while (n < 2);

    //Helpfull for debugging
    //printf("Error n: %d\n", n);
    //printf("Error secon: %d\n", response[1]);

    return response[0] + 256*response[1];
}

/**
* This function is responsible for getting the current
position of a servo,
* which is identified by its channel.
*
* @param int fd - The file descriptor to the device
* @param unsigned char channel - The channel number
represented in 8 bit binary
*
* returns  - The collation of two bytes as one single number.
*/
int maestroGetPosition(int fd, unsigned char channel)
{
    unsigned char command[] = {0xAA, 0xC, 0x10, channel};
    if(write(fd, command, sizeof(command)) == -1)
    {
        perror("error writing");
        return -1;
    }

    int n = 0;
    char response[2];
    do
    {
        int ec = read(fd, response+n, 1);
        if(ec < 0)
        {
            perror("error reading");
            return ec;
        }
        if (ec == 0)
        {
            continue;
        }
```

```
        n++;

    } while (n < 2);

    return response[0] + 256*response[1];
}

/**
* This function writes a new target to a given servo channel.
*
* @param int fd - The file descriptor to the device
* @param unsigned char channel - The channel number
represented in 8 bit binary
* @param unsigned short target - We can represent two bytes in
a unsigned
* short target.
*
* returns 0 if successful or -1 if fails.
*/
int maestroSetTarget(int fd, unsigned char channel, unsigned
short target)
{
    unsigned char command[] = {0xAA, 0xC, 0x04, channel,
target & 0x7F, target >> 7 & 0x7F};
    if (write(fd, command, sizeof(command)) == -1)
    {
        perror("error writing");
        return -1;
    }
    return 0;
}

/*
* Open Device,
* Clear errors if any
* Get and print current position
* Set and print current position given the last position
*
* @returns int - Returns -1 on error, otherwise 0
*/
int open_device()
{
    // Open the Maestro's virtual COM port.
    const char * device = "/dev/ttyAMA0"; // Linux
    int fd = open(device, O_RDWR | O_NOCTTY);

    struct termios options;
    tcgetattr(fd, &options);
    cfsetispeed(&options, B9600);
    cfsetospeed(&options, B9600);

    options.c_cflag &= ~PARENB;
    options.c_cflag &= ~CSTOPB;
    options.c_cflag &= ~CSIZE;
```

```
    options.c_cflag |= CS8;

    // no flow control
    options.c_cflag &= ~CRTSCTS;

    options.c_cflag |= CREAD | CLOCAL; // turn on READ &
ignore ctrl lines
    options.c_iflag &= ~(IXON | IXOFF | IXANY); // turn off
s/w flow ctrl

    options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); //
make raw
    options.c_oflag &= ~OPOST; // make raw

    // see: http://unixwiz.net/techtips/termios-vmin-
vtime.html
    options.c_cc[VMIN] = 0;
    options.c_cc[VTIME] = 20;

    if (tcsetattr(fd, TCSANOW, &options) < 0)
    {
        perror("init_serialport: Couldn't set term
attributes");
        return -1;
    }

    if (fd == -1)
    {
        perror(device);
        return -1;
    }

    return fd;
}
```

**Pololu.h**

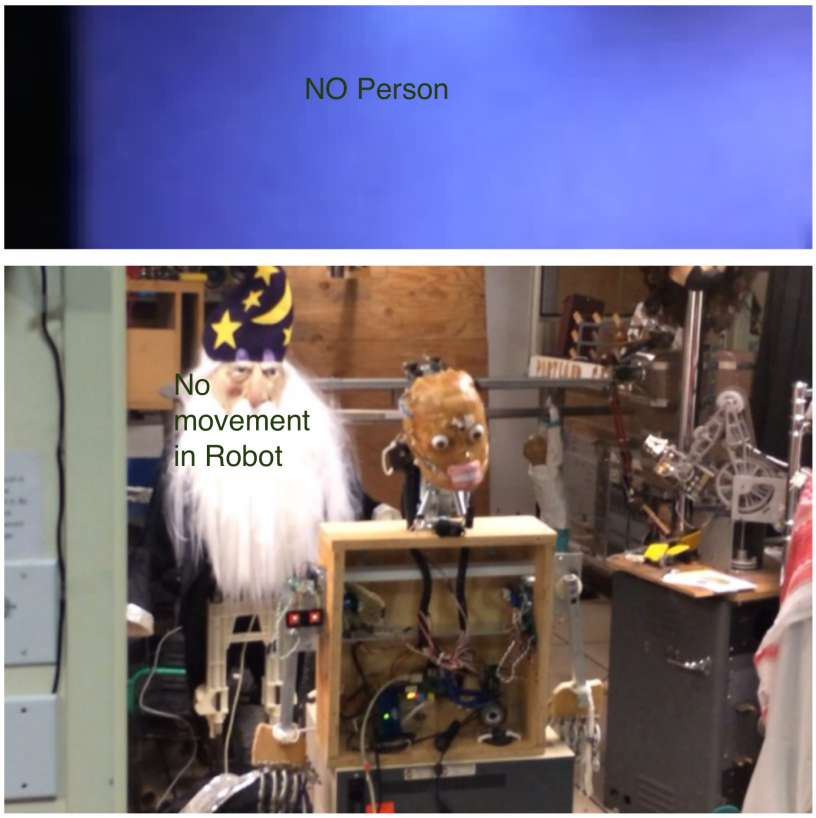
```
/**
 * The header file for pololu.cpp
 */

/* error bits per section 4b of Pololu Mini Maestro User Guide
*/
#define SERIAL_SIGNAL_ERROR 1 << 0;
#define SERIAL_OVERRUN_ERROR 1 << 1;
#define SERIAL_RX_BUFFER_FULL 1 << 2;
#define SERIAL_CRC_ERROR 1 << 3;
#define SERIAL_PROTOCOL_ERROR 1 << 4;
#define SERIAL_TIMEOUT_ERROR 1 << 5;
#define SCRIPT_STACK_ERROR 1 << 6;
#define SCRIPT_CALL_STACK_ERROR 1 << 7;
#define SCRIPT_PROGRAM_COUNTER_ERROR 1 << 8;
#define POLOLU_FIRST_CHANNEL 0        /* lowest number pololu
channel */
```

```
#define POLOLU_LAST_CHANNEL 23         /* highest number pololu
channel */


/* function declarations */
int open_device();
int maestroGetError(int fd);
int maestroGetPosition(int fd, unsigned char channel);
int maestroSetTarget(int fd, unsigned char channel, unsigned
short target);
int maestroSetSpeed(int fd, unsigned char channel, unsigned
short speed);
int maestroSetAcceleration(int fd, unsigned char channel,
unsigned short acceleration);
int maestroGetMovingState(int fd);
int maestroSetAllTargets(int fd, unsigned short
target_array[]);
```

**RESULTS:**




No person in the Kinect range, hence there is no movement in the robot arm


Person in the Kinect range, Facedetect program detects the person with red square displayed on the output screen and the movement in the robot arm is see

**FUTURE IMPROVEMENTS:**

The following are some suggested improvement can be done in future:

- Hardware functionality to the fingers and head movement can be done.
- Wifi support to the raspberry pi
- At present the there is a initial lag of 15 seconds from person detection to Sonbi action which can be further optimized by using gray images and reducing the pixel rate of the output image.
- A lot funny actions and movements can be added to the Sonbi