Robotic Arm Control using Qualcomm RB3

Sahaj Sarup
Applications Engineer, Linaro-96Boards

Linaro connect
San Diego 2019

Contents

- Overview
- What is an RB3?
- Robotic Arm project overview.
- Design decisions.
- Project Breakdown
  - Hardware Requirements
  - Software Requirements
  - Vision
  - Voice
  - Motion
  - Bringing it all together
- Special Thanks!
- Resources

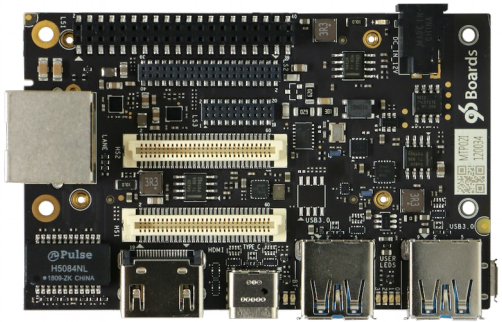
Overview

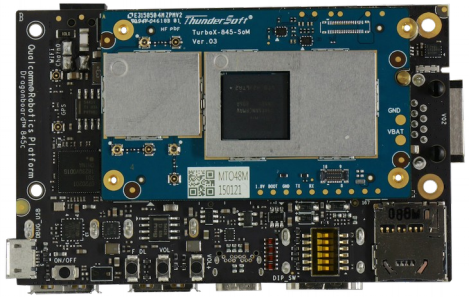
- Purpose of this project:
  - Introducing developers to the RB3 and SDA845 as a Robotics Platform
  - Show capabilities of RB3 in the field of:
    - Near Real-Time Vision Processing: Using OpenCV
    - User Voice Input: Using Python’s SpeechRecognition Library
    - Robotics: Controlling a 6DOF Robotic Arm
  - Serve as a sample project

- Target Audience
  - People in the field of Robotics who need something with a lot more oomph than a Pi.

Meet the RB3

- CPU: Armv8
  - 4x Kryo 385 Gold 2.8 GHz
  - 4x Kyro 385 Silver 1.7GHz
- GPU:
  - Adreno™ 630 GPU
  - OpenGL™ ES 3.2
- DSP: Hexagon™ 685 DSP
- RAM: 4GB LPDDR4x SDRAM @ 1866 MHz
- Storage:
  - 64GB UFS 2.1 on-boardstorage
  - 1 x MicroSD card slot





- Expansion Interface:
  - **HS1**:1 x 60 pin High-Speed connector (4L-MIPI DSI, USB 2.0 x2,I2C x2, 2L+4L-MIPI CSI)
  - **HS2**:1x 60 pin High-Speed connector (4L-MIPI CSI x 2, SSC SPI, PCIe 3.0,USB 3.0 x1, GPIO x 9)
  - **LS1**:1x 96boards 40 pin Low-Speed connector (UART x2, SPI, I2S, I2C x2, GPIO x12, DC power)
  - **LS2**:1x 96boards 40 pin Low-Speed connector(headset, stereo speaker, DMIC I/F x3, CAN, I2S, GPIO x 7, PWM x 2, ADC x 2)
  - **LS3**:1x 96boards 20 pin Low-Speed connector (SSC SPI x 3, SSC I2C, sensor interrupt x 5)

Project Overview

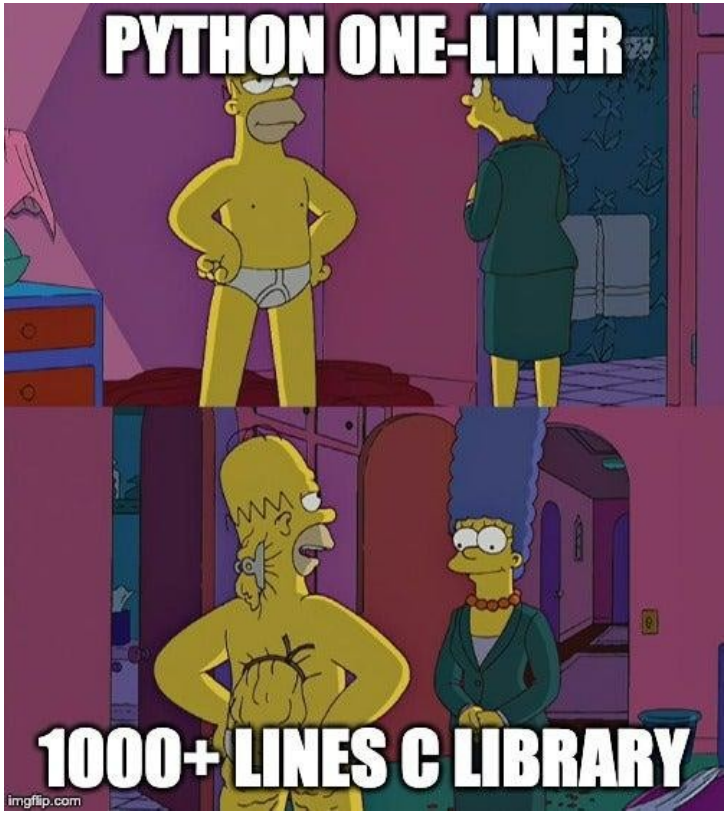
- Basic Idea of Operation
  - The user says a trigger phrase, followed by an action command
    - “Hey July”
    - “Pick up the red ball”
  - RB3 understands the user’s voice input and separates it into:
    - Color: red
    - Shape: Circle
    - Action: Pickup
  - RB3 detects the required object using OpenCV with input from camera mounted on the arm.
  - Once the object is found, it maneuvers the arm to grab and pickup the said object.

Design Decisions

- Elephant-duino in the room:
  - Used to control the i2c based PCA9685 PWM Driver
  - Arduino being used as a stop-gap solution till correct firmware for SPI/i2c IP block are available from vendor.
  - The exact same arduino code-logic can be ported to python running directly on the RB3

- Python
  - Pros
    - Extensive educational and sample content around OpenCV
    - SpeechRecognition Library
  - Cons:
    - GIL: Global Interpreter Lock
      - https://realpython.com/python-gil/

Graphical Representation of OpenCV and NumPy achieving parallelism in Python



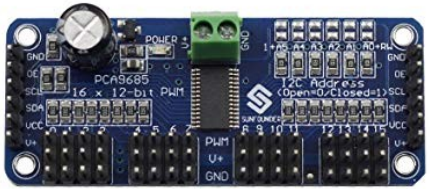
Design Decisions Cntd...

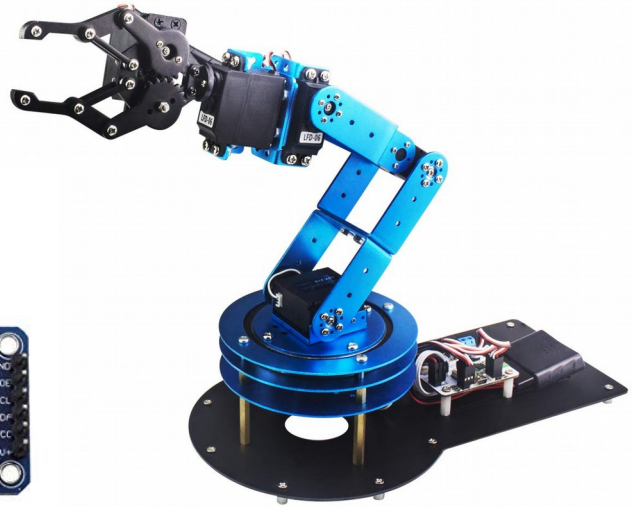
- Logitech C922
  - Fast autofocus
  - Auto White balance: consistent contours in different lighting conditions
- OpenCV 3.2
  - OpenCV 4.x is buggy when using with Aarch64 Hardware
    - Issues with libgomp: https://github.com/opencv/opencv/issues/14884

Project Breakdown

Hardware Requirements

- RB3
  - Optionally Arduino UNO
- 6DoF Robotic Arm
  - I used the LewanSoul 6DOF Robotic Arm Kit
  - Some 5DoF work as well
- Logitech C922 or comparable USB Webcam
- PCA9685 Servo Control Board
  OR
  RB3 Robotics Mezzanine: TBA



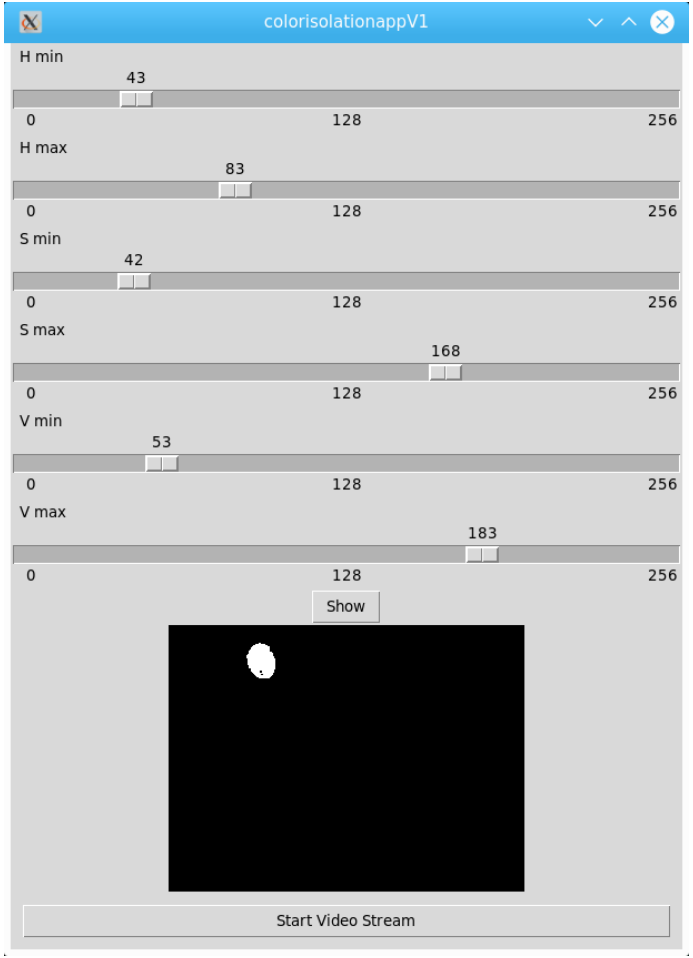


Software Requirements

- Linaro Debian Build for RB3:
  - https://www.96boards.org/documentation/consumer/dragonboard/dragonboard845c/downloads/debian.md.html
- Arduino IDE installed on RB3
- Run the install-opencv.sh script to install OpenCV and other dependencies

Vision - OpenCV

- Setting up contours
  - **Contours** are defined as the line joining all the points along the boundary of an image that are having the same intensity.
  - Intensity is defined by HSV colorspace, the values for which have to be set manually on a per color basis.
  - These help us sort object by color and later sort them by shape.



Vision - OpenCV

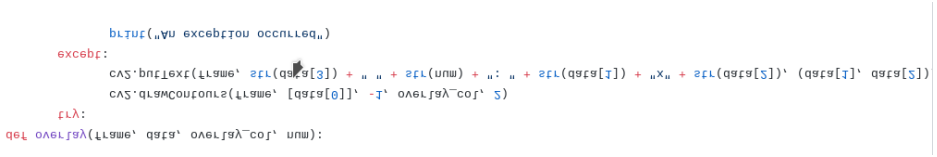
- Detecting Shape
  - **Prepare the frame**
    - Blur the entire frame
    - Convert from BGR to HSV
    - Perform a series of dilations and erosions to remove any small blobs left in the mask
    - Find contours in the mask and initialize the current (x, y) center of the ball
  - **Detect Shape:**
    - Using arcLength calculate contour perimeter.
    - Using approxPolyDP calculate the number of curvers
    - Assign shape as per the number of curves calculated.
  - Return contour, Xpos, YPos and shape values as a list

```
def detects(frame, lower, upper):
        # resize the frame, blur it, and convert it to the HSV
        # color space
        blurred = cv2.GaussianBlur(frame, (11, 11), 0)
        hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

        # construct a mask for the color "green", then perform
        # a series of dilations and erosions to remove any small
        # blobs left in the mask
        mask = cv2.inRange(hsv, lower, upper)
        mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)

        # find contours in the mask and initialize the current
        # (x, y) center of the ball
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)


        data = [0,0,0,0]
        data_arr=[data]

        if len(cnts) > 0:
                c = max(cnts, key=cv2.contourArea)
                sd = ShapeDetector()
                i = 0
                for c in cnts:
                        ((x, y), radius) = cv2.minEnclosingCircle(c)
                        M = cv2.moments(c)
                        cX = int((M["m10"] / M["m00"]))
                        cY = int((M["m01"] / M["m00"]))
                        shape = sd.detect(c)
                        data = [c, cX, cY, shape]
                        data_arr.insert(i, data)
                        i = i + 1

        return data_arr
```

```
def detects(frame, lower, upper):
        # resize the frame, blur it, and convert it to the HSV
        # color space
        blurred = cv2.GaussianBlur(frame, (11, 11), 0)
        hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

        # construct a mask for the color "green", then perform
        # a series of dilations and erosions to remove any small
        # blobs left in the mask
        mask = cv2.inRange(hsv, lower, upper)
        mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)

        # find contours in the mask and initialize the current
        # (x, y) center of the ball
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)


        data = [0,0,0,0]
        data_arr=[data]

        if len(cnts) > 0:
                c = max(cnts, key=cv2.contourArea)
                sd = ShapeDetector()
                i = 0
                for c in cnts:
                        ((x, y), radius) = cv2.minEnclosingCircle(c)
                        M = cv2.moments(c)
                        cX = int((M["m10"] / M["m00"]))
                        cY = int((M["m01"] / M["m00"]))
                        shape = sd.detect(c)
                        data = [c, cX, cY, shape]
                        data_arr.insert(i, data)
                        i = i + 1

        return data_arr
```

Vision - OpenCV

- Adding Visual overlay
  - Use drawContours to draw an outline over the object in the frame
  - Use putText to add text such as
    - Shape
    - X,Y Position with reference to current frame
    - Object count in case of multiple objects with same color and shape
  - Note: This is purely a eye candy step and has no effect on the actual working of the Robotic Arm.



Voice - SpeechRecognition Library

- Basic Detection
  - Using system default mic as source
  - Use adjust_for_ambient_noise for noise cancellation
  - listen(source) for audio input
  - recognize_google(audio) to do speech recognition using Google Speech API, aka Web Speech API
    - Not to be confused with Google Cloud Speech API

```
def detect():
	with mic as source:
		#wait for a second to let the recognizer adjust the
		#energy threshold based on the surrounding noise level
		r.adjust_for_ambient_noise(source)
		print("Say Something")
		#listens for the user's input
		audio = r.listen(source)

		try:
			text = r.recognize_google(audio)
			print("you said: " + text)
			return text

		#error occurs when google could not understand what was said

		except sr.UnknownValueError:
			print("Google Speech Recognition could not understand audio")
			return 1

		except sr.RequestError as e:
			print("Could not request results from Google Speech Recognition service; {0}".format(e))
			return -1
```

Voice - SpeechRecognition Library

- Crude Language Processing
  - Using difflib
  - Using the speech to text string provided by SR Libraray, apply closeMatches and compare against predetermined lists to get
    - Action
    - Color
    - Shape
  - Return as a list [action, color, shape]

```
def run():
    if (detect() == "hey July"):
        print("what do you want?")
        instruction = detect()
        if (instruction != 1 or instruction != -1):
            action = closeMatches(action_pattern, instruction)
            if action != 1:
                print("Action: " + action)
            else:
                not_understood()
                return 0
            color = closeMatches(color_pattern, instruction)
            if color != 1:
                print("Color: " + color)
            else:
                not_understood()
                return 0
            obj = closeMatches(obj_pattern, instruction)
            if obj != 1:
                print("Object: " + obj)
                voice_dat = [action, color, obj]
                return voice_dat
            else:
                not_understood()
                return 0
        else:
            not_understood()
            return 0
    else:
        not_understood()
        return 0
```

Motion - Arduino

- Adafruit's PWMServoDriver library to drive the Servos using a PCA9685 16 channel PWM driver.
- MIN / MAX / DEFAULT_PULSE_WIDTH and FREQUENCY are specific to the servos used, make sure to read that datasheet for the servos that you intend to use.
- Integers i, j and k are declared with default position values for the serve, these value will later be used to track the servos.
- Convert Degrees to PWM Values for servos.

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

#define MIN_PULSE_WIDTH 800
#define MAX_PULSE_WIDTH 2200
#define DEFAULT_PULSE_WIDTH 1500
#define FREQUENCY 50

uint8_t servonum0 = 0;
int i = 90, j = 30, k = 90;
char a;

void setup()
{
Serial.begin(115200);
pwm.begin();
pwm.setPWMFreq(FREQUENCY);
pwm.setPWM(2, 0, pulseWidth0(90));
pwm.setPWM(1, 0, pulseWidth0(90));
pwm.setPWM(0, 0, pulseWidth0(i));
pwm.setPWM(3, 0, pulseWidth0(j));
pwm.setPWM(5, 0, pulseWidth0(0));
}
int pulseWidth0(int angle)
{
int pulse_wide, analog_value;
pulse_wide = map(angle, 0, 180, MIN_PULSE_WIDTH, MAX_PULSE_WIDTH);
analog_value = int(float(pulse_wide) / 1000000 * FREQUENCY * 4096);
//Serial.println(analog_value);
return analog_value;
}

int pulseWidth3(int angle)
{
int pulse_wide, analog_value;
pulse_wide = map(angle, 0, 180, 1000, 2000);
analog_value = int(float(pulse_wide) / 1000000 * FREQUENCY * 4096);
//Serial.println(analog_value);
return analog_value;
}
```

Motion - Arduino

- Since the arduino is communicating to the RB3 over USB-UART and shows up as a tty device on the RB3, so we enable Serial at 115200 baud
- Depending upon what character is sent over uart, position of a specific servo(s) is incremented or decremented by one degree.
- There are also commands to open and close the claw, as well as reset the position of the arm.
- This is basically how the RB3 orders the Arduino to move the Robotic Arm.

```
if(Serial.available())
{
 a = Serial.read();

   if (a == 'a' && i != 165)
 {
   pwm.setPWM(0, 0, pulseWidth0(i));
   i++;
   //delay(200);
 }
 else if (a == 'd' && i != 0)
 {
   pwm.setPWM(0, 0, pulseWidth0(i));
   i--;
   //delay(200);
 }
 else if (a == 'w' && j != 165)
 {
   pwm.setPWM(3, 0, pulseWidth0(j));
   j++;
   //delay(200);
 }
  else if (a == 's' && j != 0)
 {
   pwm.setPWM(3, 0, pulseWidth0(j));
   j--;
  // delay(200);
 }

   else if (a == 'r' && k != 160)
 {
   pwm.setPWM(2, 0, pulseWidth0(k));
   pwm.setPWM(1, 0, pulseWidth0(k));
   k++;
   if ( k >= 159 )
   {
     Serial.print('s');
   }

   //delay(100);
 }
  else if (a == 'f' && k != 0)
 {
   pwm.setPWM(2, 0, pulseWidth0(k));
   pwm.setPWM(1, 0, pulseWidth0(k));
   k--;
```

Motion - RB3 (Future Plans)

- Most of the logic from arduino code is reusable, just needed to be converted to python.
- Adafruit’s PWMServoDriver library will be replaced by MRAA/UPM’s Servo library that is also compatible with PCA9685
- This will be controlled by a separate python program.
- All the UART functionality gets converted to memcached for a similar effect.

Bringing it all together!

- Due to Python's GIL
  - We have a separate object detections script called shape.py
  - And a main.py for voice and motion
  - These scripts share data using memcahed

- Memcached: pymemcache
  - Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

- Pymemcached can only store string values.
  - All the data transferred is in lists.
  - It is converted to string using json.dumps
  - And converted back to lists using json.loads

Bringing it all together!

Shape.py

- All the detected shapes are stored in a nested list

  shape_data = [shape_blue_data,
  shape_green_data, shape_red_data]

- Each member of list is another list that contains

  [Xpos, YPos, shape]

- Convert list to string using json.dumps

- Share string using memcahed

```
shape_data = [shape_blue_data, shape_green_data, shape_red_data]
shape_data_str = json.dumps(shape_data)
client.set('vision_data', shape_data_str)
```

Bringing it all together!

main.py

- Get string using memcahed
- Convert string to lists using json.load
- Match data in list to data received from user input.
- Get X&Y positional data
  - move the arm left or right so that the Xpos of the object falls approximately at the center of the frame.
  - do the same for Ypos by moving the wrist forward and backwards
  - Now the object should be approximately in the middle of the screen.
- Start lowering the Arm till it reaches a predetermined distance from the ground
- Said predetermined value is hardcoded in the arduino code
- Perform pick and place.

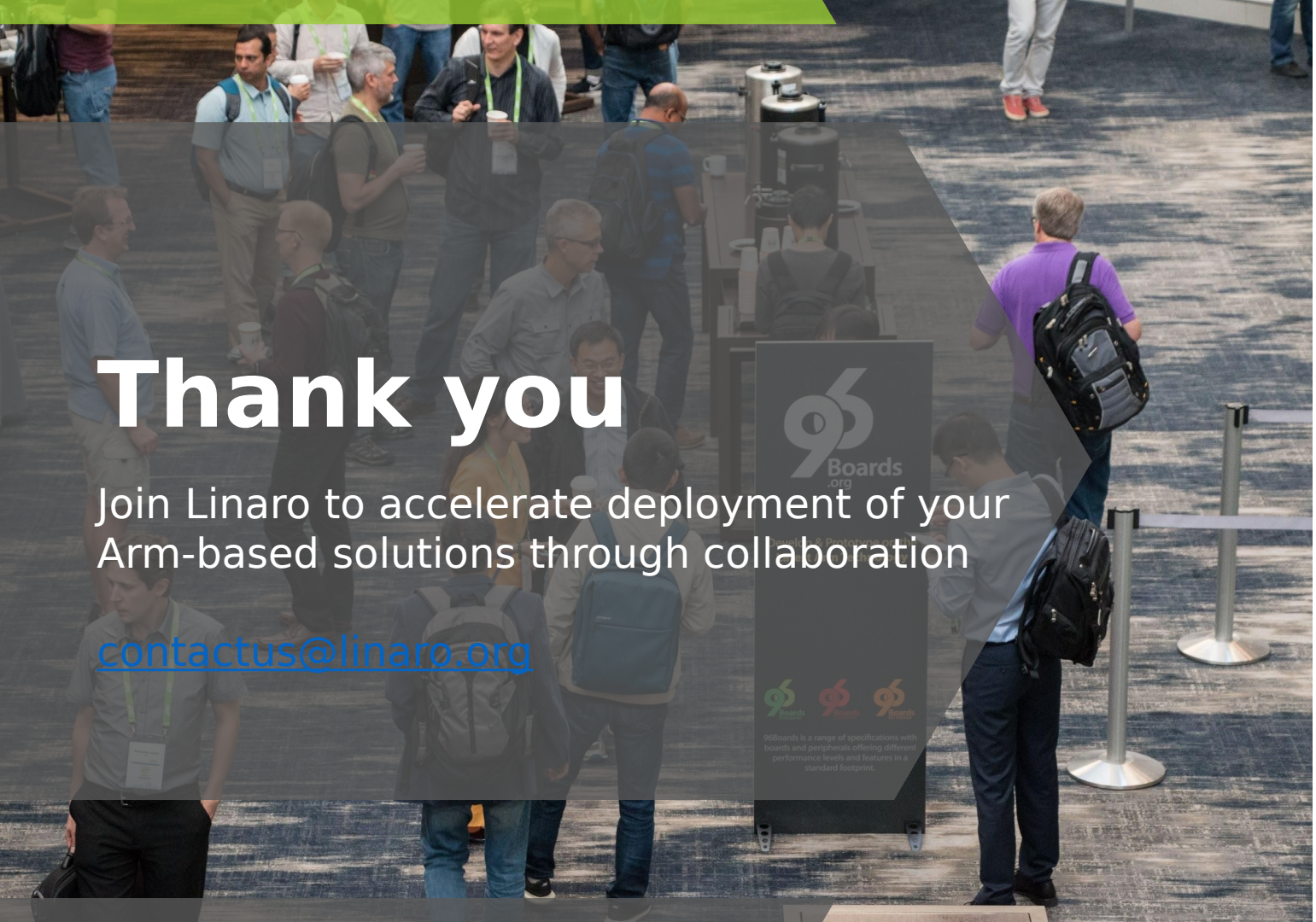
```
if(shape_data[col][0][2] == voice_data[2]):

            # Align the arm so that the X pos of the object falls in the center of the frame
    while ( ( ( loca[0] >= ((600/2)+10) ) or ( loca[0] <= ((600/2)-10) ) ) ):
        print("Required Object at X:" + str(loca[0]) + " Y: " + str(loca[1]))
        shape_data_str = client.get('vision_data')
        shape_data = json.loads(shape_data_str)
        loca[0] = shape_data[col][0][0]
        loca[1] = shape_data[col][0][1]
        if (loca[0] <= ((600/2)+10)):
            print("d")
            serialPort.write(str.encode('d'))
        elif (loca[0] >= ((600/2)+10)):
            print("a")
            serialPort.write(str.encode('a'))
        time.sleep(0.1)

            # Align the arm so that the Y pos of the object falls in the center of the frame
    while ( ( ( loca[1] >= ((480/2)+10) ) or ( loca[1] <= ((480/2)-10) ) ) ):
        print("Required Object at X:" + str(loca[0]) + " Y: " + str(loca[1]))
        shape_data_str = client.get('vision_data')
        shape_data = json.loads(shape_data_str)
        loca[0] = shape_data[col][0][0]
        loca[1] = shape_data[col][0][1]
        if (loca[1] <= ((480/2)+10)):
            print("s")
            serialPort.write(str.encode('s'))
        elif (loca[1] >= ((480/2)+10)):
            print("w")
            serialPort.write(str.encode('w'))
        time.sleep(0.1)
    test1=0

            # Start lowering the Arm till it reaches a predetermined distance from the ground
            # Said predetermined value is hardcoded in the arduino code
    while (serialPort.read().decode() != "s"):
        print("r")
        serialPort.write(str.encode('r'))
        time.sleep(0.1)
```

Special Thanks!

- Lauren Leung and the Qualcomm Robotics Team for creating the RB3 Platform and sponsoring this project.
- Qualcomm Landing Team for support, device enablement and Debian builds with mainline-tracking Linux Kernel
- Don Harbin and his OpenCV M&M Demo
  - https://github.com/96boards/opencv-color-tracking-demo
- PyImage Search: For various tutorials and ideas
  - https://www.pyimagesearch.com/

Resources

- WIP Github Repository:
  - https://github.com/ric96/RB3-RoboticArm
- Robotic Arm Blog Series with in-depth code explanation
  - https://www.96boards.org/blog/rb3-arm-intro/
- General documentation and resources for the RB3
  - https://www.96boards.org/product/rb3-platform/
- Reach me at
  - sahaj.sarup@linaro.org
  - irc: ric96



Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org

Linaro connect
San Diego 2019