Authors: Andrew Lin and Shaun Tran

Instructor: Prof. Faramarz Mortezaie

Course: Robotics and Embedded Systems

May 17, 2018

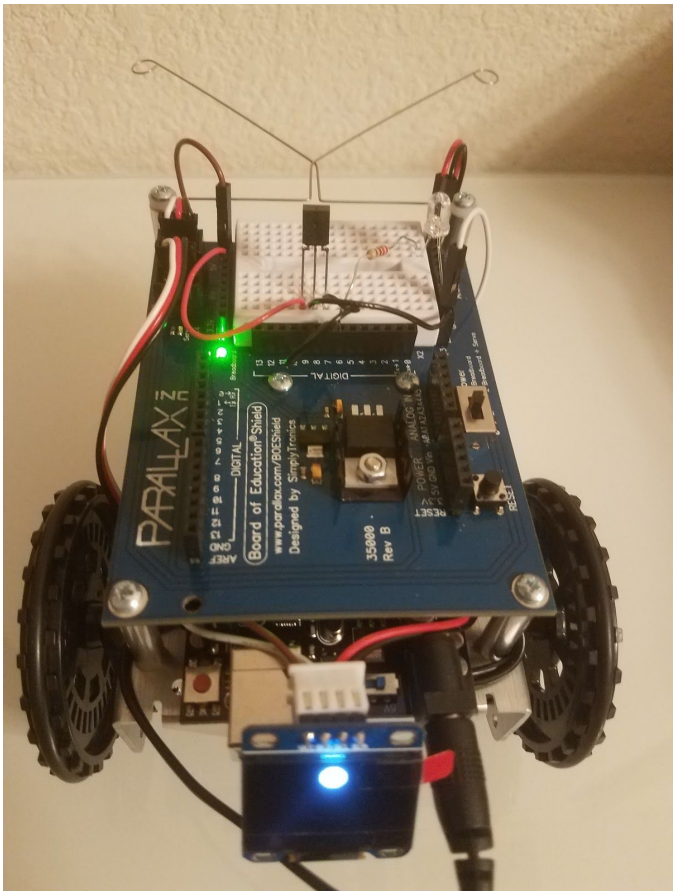
Project Report: Remote Controlling Robot Shield

Introduction:

Our project is about remote controlling a Parallax Incorporated Robot Shield that is assembled with Arduino microcontroller. Remote controlling has been a never-ending idea of many companies such as Amazon, Google, Apple, Sony, Microsoft, etc. Instead of calling this remote control, sometimes they also call this a wireless connection. Television often comes with a remote to control the TV volume, channels, and settings while we just relax sitting on the couch and pressing buttons on a remote to enjoy programs of our own TV. Gaming consoles such as Sony's Playstation and Microsoft's Xbox both abandoned the wired controller that was connected to the system by implementing a wireless controller where the player can relax without the wire of the wired controller interfering with their playing time. That is why it is the goal of our project to provide a remote controlled feature where we can control the robot from afar as long as it is clearly in our sights.

Robot:

We implemented two different types of remote to control the robot. The first one is using a TV remote and the second one is using gesture remote. By decoding the instructions from the remote, the robot will respond to the command by navigating the corresponding movement. For example, the up button instructs the robot to move forward, the down button means backward, the left button means spinning counter-clockwise while maintaining its position, similarly the right button means spinning clockwise while maintaining its position. We also decoded instructions for the robot to make left turn forward, right turn forward, left turn backward, and right turn backward. Last but not least, we also decoded a stop button so the robot can pause and resume from one movement to the next one. There is also an OLED 128x64 pixel display that shows a circle on the screen to indicate where the robot is going to move. For example in the picture above, if the circle is on the top of the screen, the robot is moving forward.



TV Remote Control:

The way the TV remote communicates with the robot is through the infrared LED transmitter on the remote and the infrared sensor receiver on top of the robot. We have used the IRremote library that supports most common TV remote protocol including NEC, Sony, Panasonic. In this project, the TVpad remote is using NEC protocol:


```
#include <IRremote.h>

#define RECV_PIN 7
IRrecv irrecv(RECV_PIN);
irrecv.enableIRIn();

decode_results results;
if (irrecv.decode(&results)){
      switch (results.decode_type){
        case NEC: Serial.print("NEC"); break ;
        case SONY: Serial.print("SONY"); break ;
        case RC5: Serial.print("RC5"); break ;
        case RC6: Serial.print("RC6"); break ;
        case DISH: Serial.print("DISH"); break ;
        case SHARP: Serial.print("SHARP"); break ;
        case JVC: Serial.print("JVC"); break ;
        case SANYO: Serial.print("SANYO"); break ;
        case MITSUBISHI: Serial.print("MITSUBISHI"); break ;
        case SAMSUNG: Serial.print("SAMSUNG"); break ;
        case LG: Serial.print("LG"); break ;
        case WHYNTER: Serial.print("WHYNTER"); break ;
        case AIWA_RC_T501: Serial.print("AIWA_RC_T501"); break ;
        case PANASONIC: Serial.print("PANASONIC"); break ;
        case DENON: Serial.print("DENON"); break ;
       default:
        case UNKNOWN: Serial.print("UNKNOWN"); break ;
       }
}
```

The robot will decode instructions from TVpad remote buttons and respond with nine possible movements as defined by ir_command:

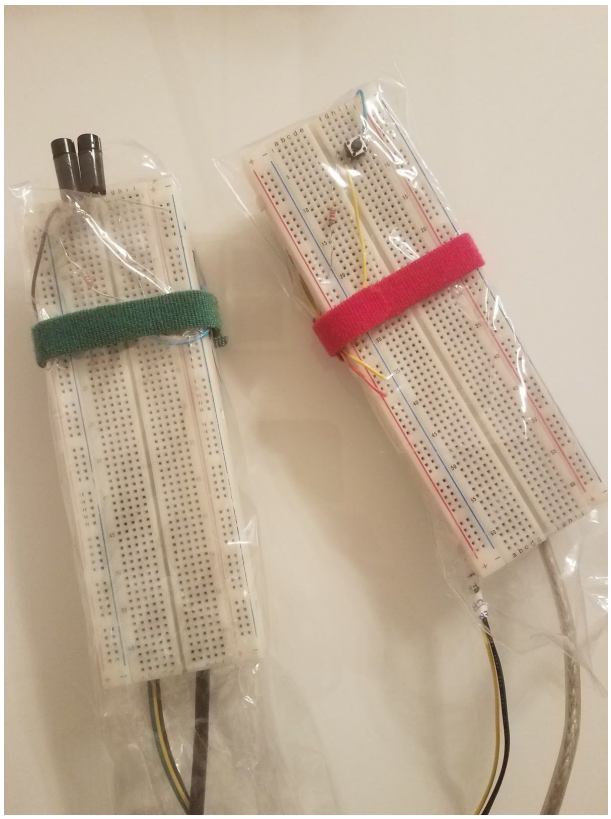
```
enum ir_command {
        Undefined, Forward, Reverse, Stop,
        LeftSpin, RightSpin,
        LeftForward, RightForward,
        LeftReverse, RightReverse,
        Record, Replay1, Replay2, Replay3, Replay4};
```


```
  switch (results.value) {
    case 0xFFAA55: ir_key = Forward; by = 10; break;
    case 0xFF629D: ir_key = Reverse; by = display.height()-10; break;
    case 0xFF6A95: ir_key = Stop; break;
    case 0xFF5AA5: ir_key = LeftSpin; bx = 10; break;
    case 0xFF4AB5: ir_key = RightSpin; bx = display.width()-10; break;
    case 0xFF9A65: ir_key = LeftForward; bx = 10; by = 10; break;
    case 0xFF42BD: ir_key = RightForward; bx = display.width()-10; by = 10; break;
    case 0xFF52AD: ir_key = LeftReverse; bx = 10; by = display.height()-10; break;
    case 0xFF8A75: ir_key = RightReverse; bx = display.width()-10; by = display.height()-10; break;
    case 0xA1B79867: ir_key = Record; break;
    case 0xFF12ED: ir_key = Replay1; break;
    case 0xFF32CD: ir_key = Replay2; break;
    case 0xFF22DD: ir_key = Replay3; break;
    case 0xFF02FD: ir_key = Replay4; break;
    default: ir_key = Undefined; break;
  }
```

Gesture Remote Control:

Another way of controlling the robot is through the hand gesture. An accelerometer is a sensor device that measures an acceleration of an object and is commonly used for hand gesture application. We have implemented gesture remote control using two different Arduino devices connected by I2C wires SDA, SCL, GND.

In the picture below, the first device is Arduino 101 connected as I2C Master to recognize hand gesture (on the right with red tape). The second device is Arduino UNO connected as I2C Slave to function as infrared LED transmitter to communicate with the robot (on the left with green tape).



For example, when the hand gesture is moving up, the robot will respond by moving forward. When the hand gesture is moving down, the robot will respond by moving backward. Based on the hand gesture of Arduino 101 as I2C Master, the Arduino UNO as I2C Slave will transmit the corresponding instruction to the robot using pair of two infrared LED transmitter for better range and accuracy of transmission through line of sight of the robot.

In the picture below, the top device is Arduino UNO as I2C Slave with slave address 4 and two infrared LED which must be connected to pin3 of Arduino UNO to send the instruction to the robot using NEC protocol:


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

IRsend irsend;
Wire.begin(4);
Wire.onReceive(receiveEvent);

void receiveEvent(int howMany)
{
 data = Wire.read();
 new_data = true;
}
```

```
  char *gesture = "Undefined";
  switch (copy_data) {
   case Forward:   irsend.sendNEC(0xFFAA55,32); gesture = "Forward"; break;
   case Reverse:   irsend.sendNEC(0xFF629D,32); gesture = "Reverse"; break;
   case Stop:      irsend.sendNEC(0xFF6A95,32); gesture = "Stop"; break;
   case LeftSpin:  irsend.sendNEC(0xFF5AA5,32); gesture = "LeftSpin"; break;
   case RightSpin: irsend.sendNEC(0xFF4AB5,32); gesture = "RightSpin"; break;
   default: break;
  }
```

The bottom device is Arduino 101 as I2C Master to recognize hand gesture from the on-board accelerometer sensor and write the recognized gesture data to Arduino UNO as I2C Slave. We have used the Pattern Matching Engine (PME) library of Arduino 101 to learn from previously trained hand gesture vector stored in “ir_gesture_vector.h”

```
#include "CurieIMU.h"
#include "CuriePME.h"
#include <Wire.h>
#include "ir_gesture_vector.h"

  /* Start the IMU (Inertial Measurement Unit), enable accelerometer only */
  CurieIMU.begin(ACCEL);
  CurieIMU.setAccelerometerRate(sampleRateHZ);
  CurieIMU.setAccelerometerRange(4);

  /* Start the PME (Pattern Matching Engine) */
  CuriePME.begin();

  // I2C master sends category to I2C slave
  Wire.begin();

 // learn from previously trained hand gesture vector
 CuriePME.learn(vector_forward0, 126, Forward);
 CuriePME.learn(vector_forward1, 126, Forward);
 CuriePME.learn(vector_forward2, 126, Forward);
 CuriePME.learn(vector_forward3, 126, Forward);
 CuriePME.learn(vector_reverse0, 126, Reverse);
 CuriePME.learn(vector_reverse1, 126, Reverse);
 CuriePME.learn(vector_reverse2, 126, Reverse);
 CuriePME.learn(vector_reverse3, 126, Reverse);
 CuriePME.learn(vector_stop0, 126, Stop);
 CuriePME.learn(vector_stop1, 126, Stop);
 CuriePME.learn(vector_stop2, 126, Stop);
 CuriePME.learn(vector_stop3, 126, Stop);
 CuriePME.learn(vector_leftspin0, 126, LeftSpin);
 CuriePME.learn(vector_leftspin1, 126, LeftSpin);
 CuriePME.learn(vector_leftspin2, 126, LeftSpin);
 CuriePME.learn(vector_leftspin3, 126, LeftSpin);
 CuriePME.learn(vector_rightspin0, 126, RightSpin);
 CuriePME.learn(vector_rightspin1, 126, RightSpin);
 CuriePME.learn(vector_rightspin2, 126, RightSpin);
 CuriePME.learn(vector_rightspin3, 126, RightSpin);
```

```
 // recognize hand gesture
 readFromIMU(vector_imu);
 unsigned int gesture = CuriePME.classify(vector_imu, 126);
 if (gesture == CuriePME.noMatch) {
  Serial.println("Gesture = NoMatch!");
 } else {
  Wire.beginTransmission(4);
  Wire.write(gesture);
  Wire.endTransmission();
 }
```

The hand gesture vector is byte array of 126 in size, to be used in learning hand gesture and classifying hand gesture by the pattern matching engine of Arduino 101. Below example of four vector representing the forward hand gesture:

```
byte vector_forward0[] =
{
93,125,123,89,119,121,72,94,92,93,126,123,91,124,125,90,122,126,
91,123,124,93,126,121,92,127,120,91,125,120,91,124,119,91,123,115,
91,121,109,91,121,102,92,122,96,90,121,91,89,120,87,86,119,84,
82,119,84,80,120,83,79,123,81,76,124,82,75,122,83,76,119,79,
76,118,76,74,119,80,74,119,86,81,120,82,86,123,81,88,123,90,
93,123,91,99,122,84,99,119,91,100,120,106,110,124,103,113,122,103,
113,120,115,120,126,116,124,126,114,124,123,123,124,126,135,125,128,141};

byte vector_forward1[] =
{
92,127,126,89,109,102,72,95,93,91,126,124,91,126,124,91,127,126,
91,127,126,91,126,124,91,126,121,91,124,118,91,123,116,91,123,111,
91,123,101,91,123,92,89,122,89,86,122,88,85,122,85,82,121,81,
80,123,79,78,128,77,76,129,74,73,125,74,71,122,73,71,123,71,
72,124,74,76,123,77,81,123,78,85,122,84,91,126,90,97,128,94,
102,124,93,107,123,96,111,126,106,115,125,109,118,122,108,120,120,115,
124,124,121,126,125,122,126,123,130,127,126,136,127,129,142,125,132,157};

byte vector_forward2[] =
{
91,125,126,90,102,146,72,94,94,92,126,125,92,126,125,91,126,126,
91,125,126,92,126,125,92,126,123,91,124,122,91,123,120,91,123,117,
91,122,111,91,121,106,91,121,101,91,121,96,89,121,92,87,123,90,
85,123,87,84,122,85,82,121,84,81,121,82,80,123,83,80,122,83,
80,119,82,83,119,79,85,121,76,85,121,79,86,121,84,91,121,85,
96,121,83,97,121,86,99,120,93,104,121,96,110,122,94,112,121,95,
113,122,105,117,124,112,121,126,112,124,125,114,126,123,118,129,125,119};
```

```
byte vector_forward3[] =
{
92,126,127,93,110,117,75,95,94,92,127,125,91,126,126,90,124,127,
91,125,125,92,127,123,92,127,122,90,124,120,90,122,116,92,122,107,
93,122,98,92,121,92,90,120,89,88,121,86,85,124,84,83,127,83,
83,126,80,81,123,78,78,120,77,78,121,77,79,124,78,81,124,79,
84,123,80,89,123,80,93,124,79,96,124,82,99,124,89,103,124,91,
109,124,90,113,123,94,115,124,101,118,123,101,121,123,104,123,124,106,
125,124,109,125,125,116,126,126,119,129,126,117,131,125,118,131,124,126};
```

Observations:

The library of OLED display takes up significant amount of data memory of Arduino UNO when connected to Robot Shield and infrared sensor and may affect the performance. For example, when printing text character on the OLED display or increasing the font size, the display will blank out or even cause the robot to stop responding.

When compared to the TVpad remote which has very good range and accuracy of transmission, the infrared LED on the Arduino UNO (I2C Slave) has relatively shorter range and accuracy of 5feet from the infrared receiver.

The decision to implement gesture remote control using two Arduinos is due to:

1) IRremote library cannot compile with Arduino 101
2) Line of sight requirement for infrared transmission and sensor

That's why it becomes necessary to use two separate Arduino for gesture recognition and infrared transmission by communicating as I2C Master and Slave. In future we can try to use radio frequency (RF) or WiFi or Bluetooth for more flexible and longer range communication even over the internet.

Conclusion:

We learned how to use the infrared sensor to receive and decode instructions from either the TV remote or the Gesture remote. We also learned how to use the pattern matching engine in Arduino 101 to recognize simple hand gesture. We are surprised that we need three separate Arduinos to successfully operate the robot using the hand gesture method. Remote-controlling is still a very broad and difficult application to learn and succeed. We believe that people are probably already working hard at building all sorts of remote controlling modes of transportation such as airplanes, trains, trucks, etc.

References:

1. Infrared Light Signals

https://learn.parallax.com/tutorials/robot/shield-bot/robotics-board-education-shield-arduino/chapter-7-navigating-infrared-11

2. A Multi-Protocol Infrared Remote Library for Arduino
   http://www.righto.com/2009/08/multi-protocol-infrared-remote-library.html

3. How to setup an IR Remote and Receiver
   http://www.circuitbasics.com/arduino-ir-remote-receiver-tutorial/

4. Master Slave I2C Connection
   https://www.hackster.io/PIYUSH_K_SINGH/master-slave-i2c-connection-f1aa53

5. Complete Guide for Arduino 101
   http://cnk.makercollider.com/blog/complete-guide-for-curie-genuino-101#part2

6. Develop Application using CuriePME (Pattern Matching Engine)
   https://software.intel.com/en-us/node/720515

7. CuriePME API reference
   https://github.com/intel/Intel-Pattern-Matching-Technology#curiepmesetclassifiermode