3D accelerometer game machine

Final Report

Group 29

Siyang Xu, 460277002

Cheng Zhang, 460048837

1.Introduction

In this project, our team designed and implemented a 3D accelerometer gaming machine. The main idea is to collect the data of hand motion via accelerometers and then use the data to change the movement of ping-pong brackets. The components are: two 8*8 LED matrix board, two accelerometers, one Arduino DUE board, one Bluetooth shield and some wires. When the game starts, a signal LED light will represent the ping-pong ball and three adjacent LED lights will represent the bracket. The game needs two players. The player controls the movement of the bracket by waving the accelerometer. When the ball clicks on the bracket, it will be rebounded back towards the other player. When a player failed to hit the ball, he will lose one score. The result of the game would be updated and transmitted to the computer via Bluetooth. When the player loses five scores, he will lose the game.

2. Components & Implementation

In this section, four main components of the embedded system will be introduced in detail. For hardware connection, we first completed the power part. We connected all ground ports to the GND on Arduino board and all 5V/3.3V gate to corresponding power supply gates on Arduino DUE. Then, for the two accelerometers, we selected pin x of the accelerometer and connected it to A0 and A2 of Arduino DUE. For connections between LED matrix and Arduino DUE, Figure 6 is referenced for the implementation.

Arduino DUE board

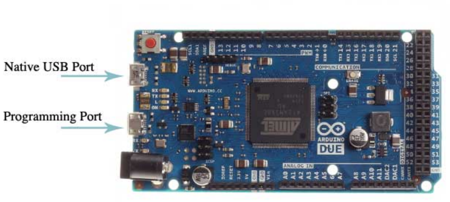

Figure 1

The Arduino DUE board is the center of this embedded system. It’s an ARM-based microcontroller with 84MHz speed and several ports for signal communication. The working voltage of Arduino DUE is 5 volts. It collects the data from the accelerometers, generating the signal to LED matrix for changing the movement of the ball and brackets on matrix. A Bluetooth shield is fixed on Arduino DUE for transmitting game results to the computer. Overall, the Arduino DUE gathers data from sensors and processes the data by its chip. Then it sends the processed data to peripherals for execution.

Bluetooth Shield

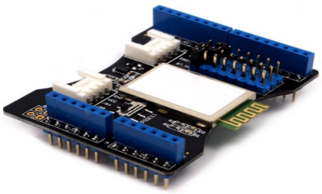

Figure 2

The model of Bluetooth shield we used is SLD63030P. The working voltage of the Bluetooth is 3.3V. In the embedded system, the Bluetooth collects the data of game results from Arduino and then emits the data to the computer. To achieve the goal, pin Rx of Arduino is connected to the pin 6 (Tx) of the Bluetooth and pin Tx of Arduino is connected to the pin 7 (Rx) of Bluetooth, then Arduino and Bluetooth can communicate with each other based on two wire connections.

Accelerometer

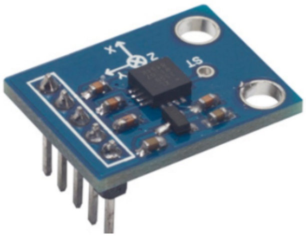

Figure 3

The accelerometer we used is ADXL335 with 5 volts driving power and 3 direction acceleration data input. The input direction used in the embedded system is the x axis. The player generates acceleration data by waving the accelerometer. Then the data will be sent to Arduino DUE via a wire connects gate x of accelerometer and the analog pin A0/A2 of Arduino DUE. Arduino DUE will process the acceleration data from the sensor and calculate its velocity, then this velocity will be applied to the bracket and the ball as the horizontal velocity.

LED Matrix

Figure 4

The part number of the LED Matrix we used is TA23-11SRWA and the working voltage is 5 volts. Since one 8*8 LED matrix is not large enough for playing ping-pong tennis, we used two LED matrixes tiled together to form a complete 8*16 LED matrix. The implementation of the LED matrixes would be the most complex part for our embedded system. The layout of the LED matrix is irregular which makes the implementation much more difficult. For connecting wires from LED matrix to Arduino DUE, we need to look at the internal diagram of LED matrix (Figure 6). Before start, we set pin31 to pin38 as R1 to R8 of the first LED matrix, pin39 to pin 46 as R1 to R8 of the second LED matrix, and pin21 to 28 as C1-C8 for both two LED matrix. For example, if we want to implement LED matrix pin 9, we first find the corresponding pin of pin 9 is R1. Since we set pin R1 as pin 31 on Arduino DUE, which means pin9 of LED matrix should be connected to pin 31 on Arduino.

3. Mechanical consideration

Since the edge of the accelerometer is very sharp and the player's hand should hold the accelerometer, for protecting the player and improving the user experience, we inserted the accelerometer to a bread board with smooth edge. For making the project more convenient to carry and play, we put the main bread board at the bottom of the black box and then use cotton wires to fasten the project. So that the project has a tight base and there is no need to take the bread board out of the black box when playing.

4. How the design works

4.1 From the player perspective

The game starts when the power cable of the Arduino board is plugged in. The ball and the brackets show up and the ball starts to move towards player 2 at a constant speed. The players can use their accelerometer controllers to control the direction and the speed of the brackets by tilting them. The steeper they are tilted, the faster do the brackets move.

The players have to do their best to make the ball land on their bracket to ensure the ball is reflected towards the opponent. Every time the ball bounces on a bracket, its vertical velocity increases a little bit to make sure the game gets more exciting as time goes on. The vertical velocity is increased automatically during the process of the game and the horizontal velocity of the ball is changed by the players. Every time the ball hits a bracket, the horizontal velocity of the bracket will be added to the horizontal velocity of the ball. Therefore, by tilting the accelerometer controller and make the bracket catch the ball while moving, the horizontal velocity of the ball can be changed dramatically in order to make it harder for the opponent to catch the ball.

When either of the players fail to catch the ball, the score board will be updated and transmitted via Bluetooth to the laptop where it is displayed. Then the ball is placed to the starting position and a new round starts with the velocity of the ball reset to the initial values.

When either player reaches 5 scores and wins, the game terminates and all LEDs are shut down. A line of text congratulating the player who wins is displayed onto the laptop.

The game machine can be restarted by pressing the reset button on the Arduino board.

4.2 From embedded system perspective

To fully understand how the game machine works, we need to take a closer look at how it is designed internally.

The 32 pins of the LED matrix are connected to the digital pins of the Arduino board correspondingly using the pin layout in Figure 5. These digitals pins are set to output mode and for pins controlling the columns, they are set to LOW by default and the digital pins controlling the rows are set to HIGH by default. By doing this, the LED matrix are ensured to remain turned off by default. To turn on the LED located at for example row 3 column 5, we need to write HIGH to digital pin 25 and LOW to digital pin 33 (see Figure 6). In this way, we can control turning one specific LED on/off.

The ball and the brackets are coded as structs in C programming language. The ball has four attributes, row, column, vertical velocity and horizontal velocity. Row and column store data of where the ball is on the LED screen and vertical and horizontal velocity store data of how fast the ball is moving along the two axes. There are two brackets and they have 3 attributes each and those are position, length and velocity. Like the ball, position determines where to place the brackets on the first row and last row respectively and velocity determines along which direction and how fast the brackets move. All these attributes of the structs continuously change using interrupt and they are constantly displayed on the LED screen using polling.

In the main loop function, we kept turning the LEDs on and off at a high frequency to solve the problem mentioned in section 6.1. By using polling to control turning the ball and brackets on we can ensure the positions of the ball and brackets are updated and displayed on time. As the attributes of the ball and brackets keep changing, we used the time counters on the Arduino to realise this functionality.

We used four timer counters with instance id TC0-TC3 to control the horizontal movement of the ball, the vertical movement of the ball, bracket number one and bracket number two respectively. All clocks are set to waveform mode, count up, RC threshold and uses TIMER_CLOCK1 whose frequency is MCK/2 = 42MHz. We then used interrupt handlers for each timer counter to execute specific tasks whenever their counter value hits the set RC value. For TC0 which controls the horizontal movement of the ball, RC value is constantly changed in the main loop according to the absolute value of the horizontal velocity of the ball. The faster the ball is, the lower the RC threshold is and the more frequently the counter value reaches this threshold and executes the incrementation of decrementation of the column value of the ball in the handler function. Whenever the ball hits the leftmost or rightmost column on the board, its horizontal velocity direction gets reversed and it will move to the opposite direction. For TC1 which controls the vertical movement of the ball, the idea of setting the handler is similar to that of TC0. The vertical velocity affects TC1's RC value and determines the frequency of counter value reaching the threshold as well as the frequency TC1 handler is called. Whenever the handler function is called, the ball's row value is incremented or decremented depending on the sign of its vertical velocity and when the ball hits the 2nd row or the 15th row, the program will compare the ball's column value to the bracket's position and length value. If the ball is within the range of the bracket, its vertical velocity direction gets reversed and the velocity of the bracket which catches the ball is added to the ball's horizontal velocity. What's more, every time the ball hits a bracket, the absolute value of its vertical velocity will be incremented twice to make the ball faster as game processes. For TC2 which controls bracket one and TC3 which controls bracket 2, their handlers are set up in a similar fashion as the previous timer counters. The only thing that needs to be mentioned is that the velocities of the brackets are set by the inputs from the accelerometer controllers. When a bracket reaches left or right edge of the screen, its velocity will be set to zero no matter what the input value from the accelerometer controller is.

The accelerometers we used were GY-61 and the voltage supplied to it is 3.3V from Arduino board. We connected their X_OUT pins to the Arduino's A0 and A2 to transmit data of how much the user is tilting the controller and then convert this data to velocity value which is transmitted to the Arduino board. However, as the RC value of TC2 and TC3 keeps changing due to the frequently varying input from accelerometer controller, we encountered problems that prevent the velocity of the brackets to vary smoothly. Therefore, the velocities of the brackets are quantized and we will discuss more in section 5.3.

We used one SeeedStudio Bluetooth Shield to communicate to our laptop wirelessly. The Bluetooth Shield is attached to the Arduino board and digital pin 6 and 7 are used for receiving and transmitting data respectively. These two pins are then connected to pin 18 and 19 as these two pins are the serial ports for USART Serial1 which we selected to use for data transmission. The Bluetooth is set and connected to an application called Bluetooth Serial Terminal on the laptop when the Arduino board is powered and every time one player scores, the locally stored data will be updated and printed onto the laptop screen providing the players with current scoreboard.

5. Design results

5.1 Parameters

For the ball, it is initialised to pop up on row 9 and column 5 with a horizontal speed of 3, in positive direction, and vertical speed of 3, the direction is determined by the round count. The brackets are initialised to be placed on column 5 with no initial velocity. As the timer clock used for all the timer counters is TIMER_CLOCK1 whose frequency is 4.2MHz, we should set RC to 4.2M if we want to call the handler once every second. The formula we use to set the RC value is $\frac{8.4*10^6}{|velocity|}$ and therefore if the initial velocity of the ball is 3, it will move one block every 0.67 seconds in real time. When the ball hits a moving bracket, their horizontal velocities will be added while taking the signs into consideration and assigned as the new horizontal velocity to the ball. For example, it the current horizontal velocity of the ball is -6 and the velocity of the moving bracket is 2, after they contact, the new horizontal velocity of the ball will be -4 and from players perspective the ball has been 'slowed down' along the horizontal axis.

5.2 Visual effect of ball movement

As the horizontal and vertical movements are separately controlled by two timer counters, they are visualized as independent movements and are then combined together to realise the movement of the ball. We tested the ball movement by giving it different velocities along the two axes and it performed the expected functionalities well. However, as our LED screen is only 16 by 8 and the LEDs are quite large in diameter, some diagonal movements at lower velocities appear to be not as smooth as expected. The horizontal and vertical movements are slightly out of phase. This will be more discussed about in section 7.1.

5.3 Bracket movement

The process of the bracket moving is User tilt the controller -> Accelerometer transmits data to Arduino -> RC value is set accordingly to this data -> Counter value reaches RC threshold and bracket position is updated. However, during testing, we encountered a problem that as the counter value kept increasing, if we changed how we tilt the controller and the RC value decreased to a value lower than current counter value, the counter would keep increasing and the handler would never be called. Therefore, we decided quantize the bracket speed to four levels, motionless, slow, medium and fast. Then we set counter value to zero every time there's a transition between two levels to avoid RC value is suddenly changed to a value less than the current counter value and the handler function will not be called. As the reading from the accelerometer varied from 400 to 620 and we wanted to set motionless speed to be 0, low speed to be 2, medium speed to be 5 and high speed to be 10, we decided to map the reading to the speed as following

Reading	400-429	430-459	460-499	500-510	511-550	551-590	591-620
Speed	-10	-5	-2	0	2	5	10

This table is obtained after many tests and is best for the player's gaming experience. When using the controllers, the users can adjust the velocity of the bracket with much ease.

5.4 Communication

Our Bluetooth was named 'Silisili' and could be successfully found and connected to on our laptop. After a delay and finishing flushing the Serial1 port, the game machine starts and data could be transmitted and received without problems.

6. Troubleshooting

While building the hardware and coding the software parts, we encountered several problems. We either solved them or made modifications to the original methods to achieve acceptable results.

6.1 LED matrix display

First thing to mention is the display problem on the LED screen. For example, if we want to turn on LEDs (1,3), (1,4), (1,5) and (2,4) simultaneously, we need to write LOW to digital pins 31 (row 1) and 32 (row 2) and HIGH to digital pins 23 (column 3), 24 (column 4) and 25 (column 5). As a result, (2,3) and (2,5) will also be turned on. This will cause undesired results when having the brackets and ball turned on at the same time. To solve this problem, we wrote code to let the brackets and ball blink alternatively. That is, brackets are turned on -> delay for 5 miliseconds -> brackets are turned off ->ball is turned on -> delay for 5 miliseconds -> ball is turned off -> brackets are turned on and so on. In this way we managed to make the brackets and the ball turn on in a simultaneous fashion with the use of persistence of vision.

6.2 Quantized bracket velocities

Another problem worth mentioning is the one we described in section 5.3. To solve this problem, we made compromise to quantize the velocity levels of the brackets in order to achieve a method that can make the game playable, sacrificing a continuous velocity change.

6.3 Pin selection

Last thing is about pin selection. When choosing which pins to use for connection between the LED matrix and the Arduino board, we initially decided to use digital pins 2-13 and A0-A3. Then after we attached the Bluetooth Shield on to the Arduino board, we realised that those pins are conflicted with those the Bluetooth needed to use. Besides that, by using A0-A3, it was harder for us to relate these pins to the corresponding rows/columns in terms of writing code. Therefore, we used digital pins 21-28 and 31-46 in the end.

7. Improvements

Although we successfully implemented the necessary functionalities of the game machine, there are a few improvements we could made to enhance the visual effects and entertaining features of this game machine.

7.1 Ball movement

As mentioned in section 5.2, the movement of the ball is not as smooth as expected at lower velocities due to too few LED dots. The changes in the horizontal and vertical movements appear to out of sync and we can observe it at low velocities. In order to solve this problem, we could use a larger LED matrix with denser LED dots. In this way, it would be harder for our eyes to observe those asynchronous movements.

7.2 Bracket movement

From section 5.3, we know that our brackets' velocities are quantized. If we could divide the reading range of accelerometer into more sections, we would be able to obtain more velocity levels and this would allow the velocity of the brackets to change with smaller difference gaps. As a result, the velocities would vary more smoothly, giving the players better gaming experience.

7.3 Special features

Because of limited time, there are several features that we planned to add to our game machine but failed to. The first one is a speaker that could play a beep sound when the ball hits a bracket and play another beep sound when a player failed to catch the ball. Another thing we wanted to implement is a 'fancy block' that would appear randomly on the screen. Whenever a player hits these blocks with the ball, advantages would be given to him such as lengthen his bracket or shorten the opponent's bracket for some time. These features acquire extra coding and would make the game more entertaining.

8.Conclusion:

Overall, we implemented the embedded system and achieved the initial design goals successfully. Looking back to the process of the project, the most challenging thing would be the logic of the code and the efforts on dealing with the handler for each component. We didn't use any C or Arduino libraries and write all codes by ourselves. We set several time counters for counting each component (ball, brackets) individually. For hardware implementation, the most difficult part would be connecting LED matrix to Arduino DUE. 32 wires were used for this implantation and they have a dense layout. For avoiding short circuit, we standardized the length of exposed copper. The test of Bluetooth would be the significant part for the embedded system. It took us some time to configure the software (Bluetooth Serial Terminal) on the computer. The whole project was time managed by the team and we completed each part before the planned time.

References:

1. Figure1: https://store.arduino.cc/usa/arduino-due

2. Figure2: http://wiki.seeedstudio.com/Bluetooth_Shield/

3. Figure3:https://www.auselectronicsdirect.com.au/arduno-3-axis-accelerometer-module?gclid=EAIaIQobChMIzMvu_q7S2wIVTIiPCh2xOQYYEAQYASABEgIxiPD_BwE

4. Figure4 http://au.element14.com/kingbright/ta23-11srwa/display-2-3-8x8-h-red-com-anode/dp/2290407?ost=TA23-11SRWA&ddkey=http%3Aen-AU%2FElement14_Australia%2Fsearch

5. Figure6:http://www.farnell.com/datasheets/1683566.pdf?_ga=2.40887650.1982811335.1526876065-1312529127.1525064512

Appendix 1:

Hardware Schematic:

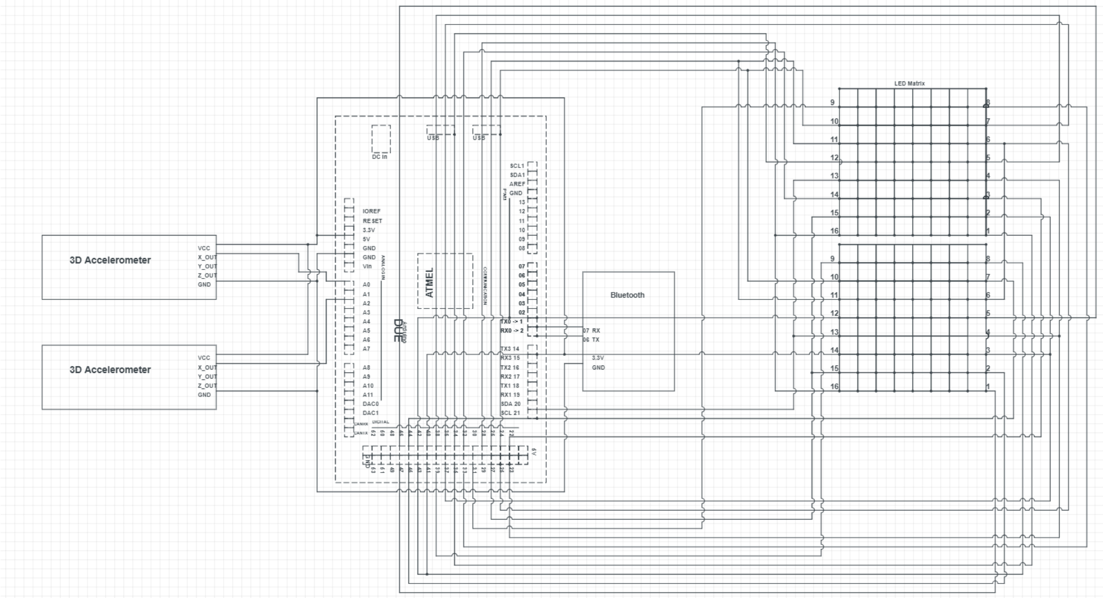

Figure 5

Package Dimensions& Internal Circuit Diagram

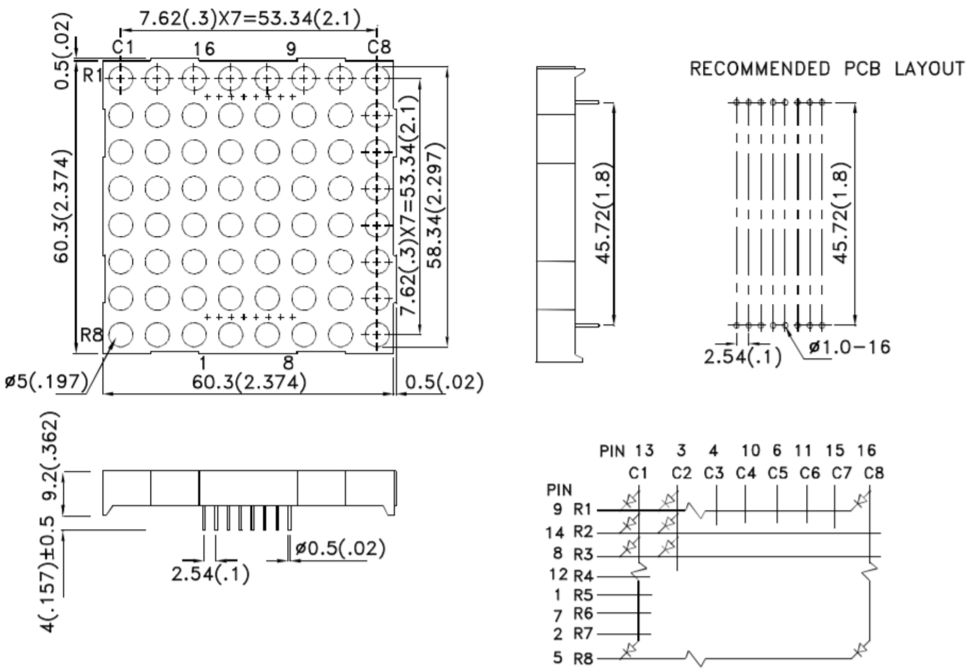

Figure 6

This is game_machine.ino (main file)

```
/*
MIT License

Copyright (c) [2018] [Siyang Xu]

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

* The University of Sydney
* School of Electrical and Information Engineering
* ELEC3607
* Program for 3d accelerometer game machine (Pong)
* Authors: Cheng Zhang 460048837
        Siyang Xu    460277002
*
* 14th June 2018
*/

#include <Slave_temperature.ino>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define  BALLX      0          // TC channel for ball horizontal
#define  BALLY      1          // TC channel for ball vertical
#define  BRACKET1   2          // TC channel for bracket_one
#define  BRACKET2   0          // TC channel for bracket_two
#define  BALLXID    ID_TC0     // Instance ID for ball horizontal
#define  BALLYID    ID_TC1     // Instance ID for ball vertical
#define  BR1ID      ID_TC2     // Instance ID for bracket_one
#define  BR2ID      ID_TC3     // Instance ID for bracket_two
#define  blueToothSerial Serial1 // Using Serial1 for Bluetooth
#define  RxD          6 // Bluetooth receive
#define  TxD          7 // Bluetooth transmit

typedef struct ball_t {
  int row;
  int col;
  double hvel;
  double vvel;
} ball_t;

typedef struct bracket_t {
  int pos;
  int row;
  int len;
  double vel;
} bracket_t;

// Allocating memory for the structs
ball_t* ball = (ball_t*)malloc(sizeof(int)*2+sizeof(double)*2);
bracket_t* bracket_one = (bracket_t*)malloc(sizeof(int)*3+sizeof(double));
bracket_t* bracket_two = (bracket_t*)malloc(sizeof(int)*3+sizeof(double));

int p1 = 0; // Player 1 score
int p2 = 0; // Player 2 score

// Sets the inital position and velocity for the ball
void initialise_ball(ball_t* ball) {
  ball->row = 9;
  ball->col = 5;
  ball->hvel = 3;
  ball->vvel = 3*pow(-1,p1+p2);
}

// Sets the intial position and velocity for the bracket
```

```
void initialise_bracket(bracket_t* bracket) {
  bracket->pos = 5;
  bracket->len = 3;
  bracket->vel = 0;
}

// Turn ball on on LED screen
void ball_on(ball_t* ball) {
  digitalWrite(47-ball->row, HIGH);
  digitalWrite(ball->col+20, LOW);
}

// Turn ball off on LED screen
void ball_off(ball_t* ball) {
  digitalWrite(47-ball->row, LOW);
  digitalWrite(ball->col+20, HIGH);
}

// Turn bracket on on LED screen
void bracket_on(bracket_t* bracket) {
  digitalWrite(bracket->pos+20, LOW);
  digitalWrite(bracket->pos+20-1, LOW);
  digitalWrite(bracket->pos+20+1, LOW);
  digitalWrite(47-bracket->row, HIGH);
}

// Turn bracket off on LED screen
void bracket_off(bracket_t* bracket) {
  digitalWrite(bracket->pos+20, HIGH);
  digitalWrite(bracket->pos+20-1, HIGH);
  digitalWrite(bracket->pos+20+1, HIGH);
  digitalWrite(47-bracket->row, LOW);
}

// When a player fails to catch the ball
void round_over() {
  ball_off(ball);
  initialise_ball(ball);
  bracket_off(bracket_one);
  initialise_bracket(bracket_one);
  bracket_off(bracket_two);
  initialise_bracket(bracket_two);
}

// When a player reaches 5 scores
// The LED screen gets turned off completely
void terminate() {
  for (int i=21;i<29;i++){
    digitalWrite(i,HIGH);
  }
  for (int i=31;i<47;i++){
    digitalWrite(i,LOW);
  }
}

void setup() {
  Serial.begin(9600);

  //Initialise the ball and the brackets
  initialise_ball(ball);
  initialise_bracket(bracket_one);
  initialise_bracket(bracket_two);
  bracket_one->row = 16;
  bracket_two->row = 1;

  // Set the pinmodes
  pinMode(RxD, INPUT);
  pinMode(TxD, OUTPUT);
  for (int i=21;i<29;i++){
    pinMode(i,OUTPUT);
    digitalWrite(i,HIGH);
  }
  for (int i=31;i<47;i++){
    pinMode(i,OUTPUT);
    digitalWrite(i,LOW);
  }

  // Establish Bluetooth connection
  setupBlueToothConnection();
  delay(1000);

  // Set timer counters and handlers
  pmc_set_writeprotect(false);

```

```
  // Timer counter for ball horizontal
  pmc_enable_periph_clk(BALLXID);
  TC_Configure(TC0, BALLX, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC | TC_CMR_TCCLKS_TIMER_CLOCK1);
  TC_SetRC(TC0, BALLX, 21000000);
  TC_Start(TC0, BALLX);
  TC0->TC_CHANNEL[BALLX].TC_IER=TC_IER_CPCS;
  TC0->TC_CHANNEL[BALLX].TC_IDR=~TC_IER_CPCS;
  NVIC_EnableIRQ(TC0_IRQn);

  // Timer counter for ball vertical
  pmc_enable_periph_clk(BALLYID);
  TC_Configure(TC0, BALLY, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC | TC_CMR_TCCLKS_TIMER_CLOCK1);
  TC_SetRC(TC0, BALLY, 21000000);
  TC_Start(TC0, BALLY);
  TC0->TC_CHANNEL[BALLY].TC_IER=TC_IER_CPCS;
  TC0->TC_CHANNEL[BALLY].TC_IDR=~TC_IER_CPCS;
  NVIC_EnableIRQ(TC1_IRQn);

  // Timer counter for bracket one
  pmc_enable_periph_clk(BR1ID);
  TC_Configure(TC0, BRACKET1, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
TC_CMR_TCCLKS_TIMER_CLOCK1);
  TC_SetRC(TC0, BRACKET1, 60000000);
  TC_Start(TC0, BRACKET1);
  TC0->TC_CHANNEL[BRACKET1].TC_IER=TC_IER_CPCS;
  TC0->TC_CHANNEL[BRACKET1].TC_IDR=~TC_IER_CPCS;
  NVIC_EnableIRQ(TC2_IRQn);

  // Timer counter for bracket two
  pmc_enable_periph_clk(BR2ID);
  TC_Configure(TC1, BRACKET2, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
TC_CMR_TCCLKS_TIMER_CLOCK1);
  TC_SetRC(TC1, BRACKET2, 60000000);
  TC_Start(TC1, BRACKET2);
  TC1->TC_CHANNEL[BRACKET2].TC_IER=TC_IER_CPCS;
  TC1->TC_CHANNEL[BRACKET2].TC_IDR=~TC_IER_CPCS;
  NVIC_EnableIRQ(TC3_IRQn);

}

// Handler for ball horizontal
void TC0_Handler() {
  TC_GetStatus(TC0, 0);
  // When the ball hits the sides
  if (ball->col==8 || ball->col == 1) {
  ball->hvel=-ball->hvel;
  }

  // When the ball hits the ends
  if (ball->row == 2) {
    if (bracket_two->pos != 1 && bracket_two->pos != 8) {
      ball->hvel += bracket_two->vel;
    }
  } else if (ball->row == 15) {
    if (bracket_one->pos != 1 && bracket_one->pos != 8) {
      ball->hvel += bracket_one->vel;
    }
  }

  // Moves the ball one step forward along its horizontal direction
  ball_off(ball);
  ball->col += ball->hvel/abs(ball->hvel);

}

// Handler for ball vertical
void TC1_Handler() {
  TC_GetStatus(TC0, 1);
  // When the ball hits the ends
  if (ball->row == 2) {
    if (ball->col < bracket_two->pos-1 || ball->col > bracket_two->pos+1) {
      p1++; // Player one scores
      print_score();
      if (p1 == 5) { // Player one wins
        blueToothSerial.println("Player 1 Wins!");
        blueToothSerial.println();
        terminate();
      }
      round_over();
      return;
    }
    ball->vvel -= 1; // Absolute value of vertical velocity gets incremented
    ball->vvel=-ball->vvel; // Vertical velocity gets reversed
  } else if (ball->row == 15) {
```

```
      if (ball->col < bracket_one->pos-1 || ball->col > bracket_one->pos+1) {
        p2++; // Player two scores
        print_score();
        if (p2 == 5) {
          blueToothSerial.println("Player 2 Wins!");
          blueToothSerial.println();
          terminate();
        }
        round_over();
        return;
      }
      ball->vvel += 1; // Absolute value of vertical velocity gets incremented
      ball->vvel=-ball->vvel; // Vertical velocity gets reversed
    }
    // Moves the ball one step forward along its veritcal direction
    ball_off(ball);
    ball->row += ball->vvel/abs(ball->vvel);
  }

  // Handler for bracket one
  void TC2_Handler() {
    TC_GetStatus(TC0, 2);
    // If the bracket has not reached the sides
    if (!((bracket_one->pos==7 && bracket_one->vel>0) || (bracket_one->pos==2&&bracket_one->vel<0))){
      bracket_off(bracket_one);
      if (bracket_one->vel <0.01 && bracket_one->vel>-0.01) { // If the velocity is 0
        return;
      }
      bracket_one-> pos += bracket_one->vel/abs(bracket_one->vel); // Move bracket one
    }
  }

  // Handler for bracket two
  void TC3_Handler() {
    TC_GetStatus(TC1, 0);
    // If the bracket has not reached the sides
    if (!((bracket_two->pos==7 && bracket_two->vel>0) || (bracket_two->pos==2&&bracket_two->vel<0))){
      bracket_off(bracket_two);
      if (bracket_two->vel <0.01 && bracket_two->vel>-0.01) {// If the velocity is 0
        return;
      }
      bracket_two-> pos += bracket_two->vel/abs(bracket_two->vel); // Move bracket two
    }
  }

  void print_score() {
    blueToothSerial.println("Current score");
    blueToothSerial.print("Player 1: ");
    blueToothSerial.println(p1);
    blueToothSerial.print("Player 2: ");
    blueToothSerial.println(p2);
    blueToothSerial.println();
  }

  void loop() {
    if (p1 != 5 && p2 != 5) { // When the game is not over
    // Using persistence of vision to turn on the ball and brackets simultaneously
    bracket_on(bracket_one);
    delay(5);
    bracket_off(bracket_one);
    bracket_on(bracket_two);
    delay(5);
    bracket_off(bracket_two);
    ball_on(ball);
    delay(5);
    ball_off(ball);

    // Classify the reading into several velocity levels
    // For bracket one
    if (analogRead(A0)<=429) {
      if (bracket_one->vel !=-10) {
        TC_Start(TC0, BRACKET1);
      }
      bracket_one->vel = -10;
    } else if (analogRead(A0)<= 459 && analogRead(A0)>=430) {
      if (bracket_one->vel !=-5) {
        TC_Start(TC0, BRACKET1);
      }
      bracket_one->vel = -5;
    } else if (analogRead(A0)<= 499 && analogRead(A0)>=460){
      if (bracket_one->vel !=-2) {
        TC_Start(TC0, BRACKET1);
```

```
      }
      bracket_one->vel = -2;
    } else if (analogRead(A0)<= 510 && analogRead(A0)>=500) {
      if (bracket_one->vel !=0) {
        TC_Start(TC0, BRACKET1);
      }
      bracket_one->vel = 0;
    } else if (analogRead(A0)<= 550 && analogRead(A0)>=511){
      if (bracket_one->vel !=2) {
        TC_Start(TC0, BRACKET1);
      }
      bracket_one->vel = 2;
    } else if (analogRead(A0)<= 590 && analogRead(A0)>=551) {
      if (bracket_one->vel !=5) {
        TC_Start(TC0, BRACKET1);
      }
      bracket_one->vel = 5;
    } else if (analogRead(A0)>=591) {
      if (bracket_one->vel !=10) {
        TC_Start(TC0, BRACKET1);
      }
      bracket_one->vel = 10;
    }

    // For bracket two
    if (analogRead(A2)<=429) {
      if (bracket_two->vel !=-10) {
        TC_Start(TC1, BRACKET2);
      }
      bracket_two->vel = -10;
    } else if (analogRead(A2)<= 459 && analogRead(A2)>=430) {
      if (bracket_two->vel !=-5) {
        TC_Start(TC1, BRACKET2);
      }
      bracket_two->vel = -5;
    } else if (analogRead(A2)<= 499 && analogRead(A2)>=460){
      if (bracket_two->vel !=-2) {
        TC_Start(TC1, BRACKET2);
      }
      bracket_two->vel = -2;
    } else if (analogRead(A2)<= 510 && analogRead(A2)>=500) {
      if (bracket_two->vel !=0) {
        TC_Start(TC1, BRACKET2);
      }
      bracket_two->vel = 0;
    } else if (analogRead(A2)<= 550 && analogRead(A2)>=511){
      if (bracket_two->vel !=2) {
        TC_Start(TC1, BRACKET2);
      }
      bracket_two->vel = 2;
    } else if (analogRead(A2)<= 590 && analogRead(A2)>=551) {
      if (bracket_two->vel !=5) {
        TC_Start(TC1, BRACKET2);
      }
      bracket_two->vel = 5;
    } else if (analogRead(A2)>=591) {
      if (bracket_two->vel !=10) {
        TC_Start(TC1, BRACKET2);
      }
      bracket_two->vel = 10;
    }

    // Constantly adjusting RC values according to the velocity values
    TC_SetRC(TC0, BALLX, 84000000/abs(ball->hvel));
    TC_SetRC(TC0, BALLY, 84000000/abs(ball->vvel));
    if (bracket_one->vel <0.01 && bracket_one->vel>-0.01) {
      TC_SetRC(TC0, BRACKET1, 84000000);
    } else {
      TC_SetRC(TC0, BRACKET1, 84000000/abs(bracket_one->vel));
    }
    if (bracket_two->vel <0.01 && bracket_two->vel>-0.01) {
      TC_SetRC(TC1, BRACKET2, 84000000);
    } else {
      TC_SetRC(TC1, BRACKET2, 84000000/abs(bracket_two->vel));
    }

    return;
    }

    // Game is over
    p1 = 0;
    p2 = 0;
    delay(10000);

```

```
}
```

This is Slave_temperature.ino
cited from
https://github.com/Seeed-Studio/Bluetooth_Shield_Demo_Code/blob/master/examples/Slave_temperature/Slave_temperature.ino

```
/*
BluetoothShield Demo Code Slave.pde. This sketch could be used with
Master.pde to establish connection between two Arduino. It can also
be used for one slave bluetooth connected by the device(PC Phone)
with bluetooth function.
2011 Copyright (c) Seeed Technology Inc.  All right reserved.
Author: Steve Chang
This demo code is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.
This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.
You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
For more details about the product please check http://www.seeedstudio.com/depot/
*/

/* Upload this sketch into Seeeduino and press reset*/

#include <SoftwareSerial.h>   //Software Serial Port
#define RxD 9
#define TxD 8

#define DEBUG_ENABLED  1

#define PIN_TEMP    A5


SoftwareSerial blueToothSerial(RxD,TxD);

int getTemp()
{
    int a = analogRead(PIN_TEMP);
    int B=3975;
    float resistance =  (float)(1023-a)*10000/a;
    float temperature = temperature=1/(log(resistance/10000)/B+1/298.15)-273.15;

    return (int)temperature;

}

void setup()
{
    Serial.begin(9600);
    pinMode(RxD, INPUT);
    pinMode(TxD, OUTPUT);
    setupBlueToothConnection();

}

void loop()
{

    char recvChar;
    while(1)
    {
        if(blueToothSerial.available())
        {//check if there's any data sent from the remote bluetooth shield
            recvChar = blueToothSerial.read();
            Serial.print(recvChar);

            if(recvChar == 't' || recvChar == 'T')
            {
                blueToothSerial.print("temperature: ");
                blueToothSerial.println(getTemp());
            }
        }
        if(Serial.available())
        {//check if there's any data sent from the local serial terminal, you can add the other applications here
            recvChar  = Serial.read();
```

```
                blueToothSerial.print(recvChar);
            }
        }



}

void setupBlueToothConnection()
{
    blueToothSerial.begin(38400);                            // Set BluetoothBee BaudRate to default baud rate 38400
    blueToothSerial.print("\r\n+STWMOD=0\r\n");              // set the bluetooth work in slave mode
    blueToothSerial.print("\r\n+STNA=SeeedBTSlave\r\n");     // set the bluetooth name as "SeeedBTSlave"
    blueToothSerial.print("\r\n+STOAUT=1\r\n");              // Permit Paired device to connect me
    blueToothSerial.print("\r\n+STAUTO=0\r\n");              // Auto-connection should be forbidden here
    delay(2000);                                             // This delay is required.
    blueToothSerial.print("\r\n+INQ=1\r\n");                 // make the slave bluetooth inquirable
    Serial.println("The slave bluetooth is inquirable!");
    delay(2000);                                             // This delay is required.
    blueToothSerial.flush();
}
```

ELEC3607/9607 Milestone IV

Project Implementation Summary Form

Category	List of items, with web link or page number where item is described
Open source compliance Are you publishing your code as open source? Explain what license you are using, or link to where you have published your project.	Yes. MIT License. Open source uploaded on Github. https://github.com/sixu3575/ELEC3607-Project (page 12)
Platforms List the platforms you have used, including processor architecture (ARM, AVR, x86, etc.), programming languages (Python, C, C++, etc.), and IDEs. If you are not using Arduino Due, justify your choice of platform.	ARM(Atmel SAM3X8E ARM Cortex-M3 CPU) Programming language: C (page 2)
Sensors and inputs List your hardware inputs (push buttons, analog sensors, I2C or SPI sensors including accelerometers, etc.) and mention on which page each input is described.	Two accelerometers (page 3)
Outputs List any mechanical, visual or other type of outputs controlled by your hardware (motors, magnets, LED, 7-Segs, LCD, etc.) and mention on which page each input is described.	Two LED Matrix(page 3) Bluetooth Shield(page 2)
Connectivity List any protocols used for communicating between your main controller and any other microcontroller excluding sensors or actuators (USART, UART over USB, Bluetooth 2, Bluetooth 4, Wi-Fi, ZigBee,	Bluetooth4 (page 2), USART 1 (page 6)

etc.) and mention on which page each input is described.	
Graphical User Interface List any GUI used for interfacing users (local LCD, Phone app, Desktop app, Web app, Web dashboard, etc.) and mention on which page each input is described.	Bluetooth Serial Terminal (a software on laptop) (page 6)
Algorithms / Logic List substantial control algorithms (state-machines, real-time operating system, filesystems, feedback algorithms, mathematical transformations, etc.) and mention on which page each input is described.	mathematical transformations (page 6) nested if conditions (page 7)
Physical case and other mechanical consideration List what casing or mechanical systems have been used in your project (3d prints, pipes, cardboard mechanics, etc.) and mention on which page each input is described.	Consideration applied for the accelerometers and the main bread board. (page 4)