CSC 460 Project 1

Spring 2019

Team 13

Hamzah Mansour	V00812108
Christina Araya	V00781339

1.0 Project Introduction

The objective of this project was to become familiarized with embedded system programming, hardware, and any hardware/software interfacing problems. This project also acts as preparation for producing a robot to play a laser game, which is yet to be specified, to be completed by the end of the term.

This project involves the following equipment:

- Two Arduino boards (ATMega2560)
- Two Bluetooth Modules
- Two PS2 Joysticks
- Two Servo Motors (pan and tilt)
- One Laser Pointer
- One Analog Light Sensor
- One LCD
- Two Breadboards
- One Power Supply

2.0 Phase 1

2.1 Description

This portion of the project required that a station be created using the materials provided excluding the Bluetooth modules. The station should control two servos, a laser, and the LCD shield as output. It should take the directions and button push of a PS2 controller joystick as input, along with the light detection provided by a light sensor. When the station is put together, the pan and tilt should be smoothly controllable using the joystick and the laser should be controlled by the push button on the joystick. The status of each directional axis of the joystick and the status of the push-button should be reflected on the LCD output, along with the status of the light sensor. If the laser is shone on the light sensor, this should be indicated on the LCD shield.

2.2 Discussion

In order to properly wire the system a block diagram was used to understand how the inputs and outputs mapped to the arduino board. The block diagram can be found in Figure 1.

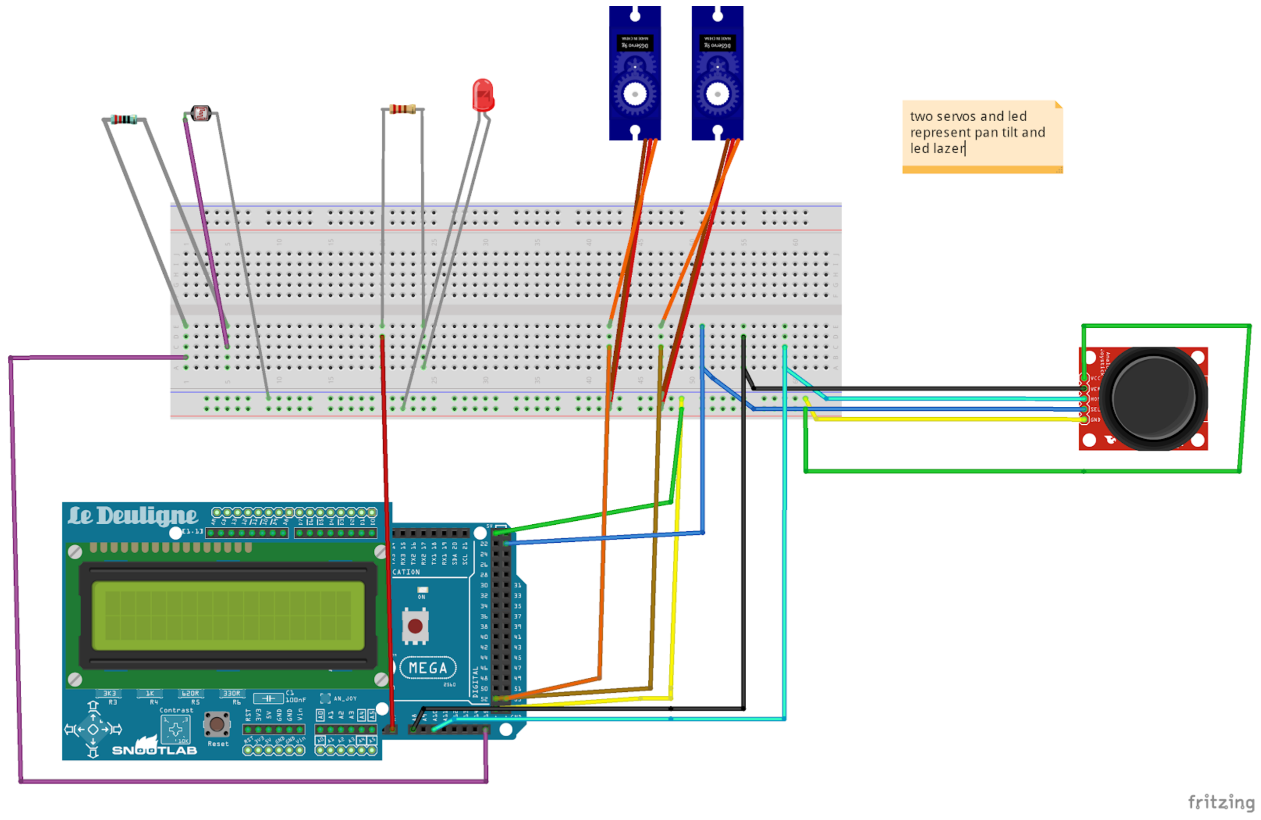

Figure 1: Phase 1 block diagram

The block diagram is colour coded by input and output associated with the various components. From the arduino board the colours are represented as following:

- Purple - light sensor component
- Red - laser component
- Black - Joystick X
- Blue - Push button
- Orange - Joystick Y
- Brown - Tilt servo
- Orange 2 - Pan servo
- Yellow - Ground
- Green - VCC

The pan and tilt kit is represented by two servos and the laser is represented by an led and resistor.

In order to regulate the speed at which the servo motors moved, a system was devised that used a timer to set the position of each servo. The tilt servo had a maximal tilt adjustment factor of 10 and the pan servo had a maximal tilt of 40 for greater speed, as it had a wider range of motion. They both sampled their relative speed from the x-axis and y-axis of the joystick. The speed of the servo was mapped from the position of the joystick to a range between negative and positive adjustment factors. The position excludes the dead band (440 - 570 for the y-axis and 425 and 575 for the x-axis) of the joystick and is linearly mapped. The speed is then limited to a smaller factor based on the previously sampled speed and added to a position which is limited to a range of 500 to 1000 for tilt and 750 and 2100 for pan. In this part a low pass filter was used on X, Y and button press (Z) inputs to reduce noise with an alpha value of 0.5. This was achieved by using exponential smoothing, as defined in equation 1 below:

$$(1)\quad s_t = \alpha x_t + (1-\alpha)s_{t-1},\ t > 0,\ 0 < \alpha < 1$$

This can then be written in a more simplified form show in equation 2:

$$(2)\quad s_t = \alpha x_t + (1-\alpha)s_{t-1} = s_{t-1} + \alpha \cdot (x_t - s_{t-1})$$

With α=0.5, this can be simplified as the following code:

```
y1 = y0 + ((y-y0)>>1);
```

Above the smoothed y1 value is computed by taking the old reading of y1, y0 and adding it to a weighted difference between the recently read y and the old value. A bitwise shift to the right (of y-y0) is equivalent to multiplying the difference by 0.5. However, z had an α-value of 0.3 to accommodate for its sensitivity.

In order to lessen the load on the servos, a timer was implemented to write the position if it had changed with the same frequency as in exercise 5, where the servo swept for a full period (360°) in about 5 seconds. However, the amount that the servo could move each update was bumped up depending on whether it was tilting (max 10 position change) or panning (max 40 position change). The position change was called the speed, in order to get the servos to update through the servo arduino library, the timers 1A,3A,4A or 5A couldn’t be used and instead 3B and 4B were used and the timers were set in the interrupts instead of using the repeater scheduler that a (1,3,4 or 5) provided.

```
// in setup cli() at beginning and sei() at end
// timer setup for tilt
  TCCR4A = 0;
  TCCR4B = 0;
  //Set prescaller to 256
  TCCR4B |= (1<<CS42);

  //Set TOP value (0.5 seconds)
  OCR4B = 1736;
  //Enable interrupt A for timer 3.
  TIMSK4 |= (1<<OCIE4B);

  TCNT4 = 0;

// interrupt handler
ISR(TIMER4_COMPB_vect){
  // routine actions here
  TCNT4 = 0;
}
```

Using the LiquidCrystal [1] Arduino library, with the LCD shield positioned on top of the arduino, the results from the light sensor output mentioned in the next paragraph, the outputs of x-axis, y-axis and push button from the thumbstick were displayed to the screen. In order to properly setup the LCD to output properly the pins for the LCD are set, and then started with set and begin functions. The LCD then will display when a cursor is set, and the print command is used. If the LCD shield is printed to with a high frequency, the output becomes difficult to read.

```
// select the pins used on the LCD panel
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

  lcd.begin(16, 2);              // start the library
  lcd.setCursor(0,0);
  lcd.print("LGHT  JSX  JSY Z");
```

The photocell (light sensor) used was connected to the base station Arduino board as shown in fig. 1, with a resistor of 6.3K. The photocell was then covered with a sealed red plastic cup to diffuse incoming light. The resistance provided by the photocell was then measured to be around 160,000Ω and 170,000Ω with the ambient light in the lab. So, a threshold of 100,000Ω was chosen. Any resistance below this threshold of 100,000Ω would indicate that a light source had targeted the photocell and this set a flag that was then used to output the state of the photocell on the LCD [2].

3.0 Phase 2

3.1 Description

With the addition of two bluetooth modules, the scenario created in phase 1 is recreated in phase 2. In this iteration however the servo motors and laser were separated onto a second arduino board. The base station arduino board should send joystick and button press commands to handle the pan, tilt, and laser to the remote board. The base station board should also handle the light sensor and LCD output, while adding time-triggered scheduling (TTS) to ensure that tasks only happen as often as they need to (setting different periods for each tasks), while still reducing CPU utilization.

3.2 Discussion

The base station as shown below is set up in a similar fashion to phase 1 just without the pan and tilt kit and with a bluetooth chip. Each of the outputs and inputs in Figure 2 correspond to a colour specified as follows:

- Purple - light sensor component
- Black - Joystick X
- Orange - Button Press
- Orange - Joystick Y
- Yellow - Ground
- Green - VCC

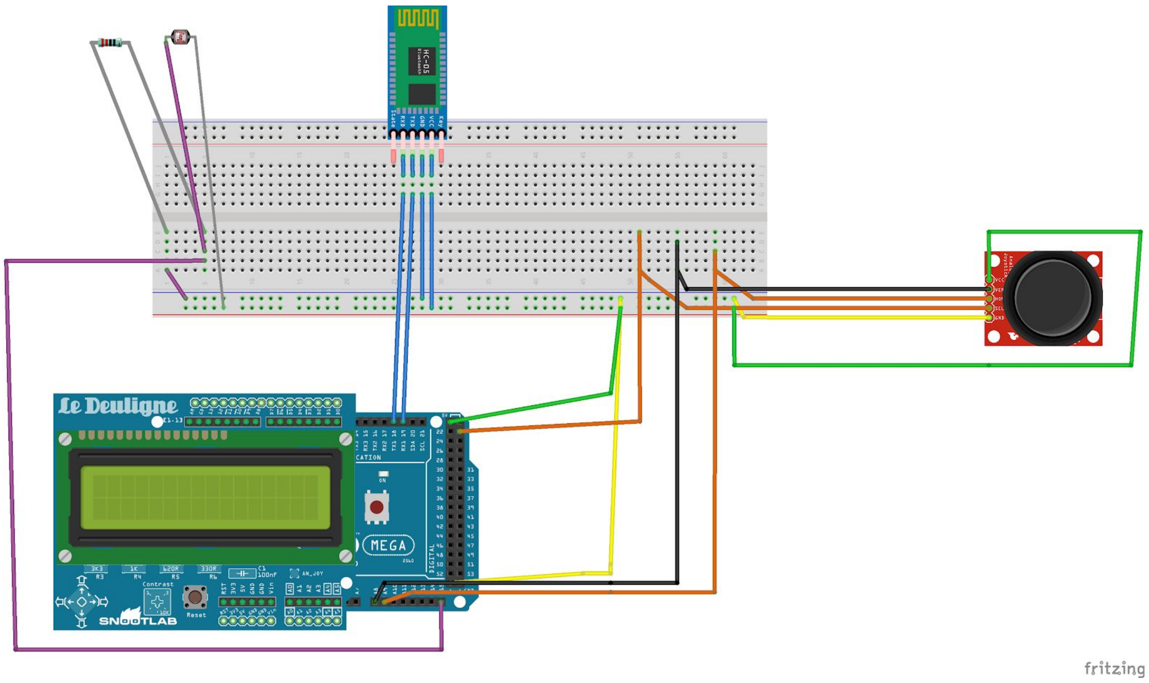

Figure 2: Phase 2 Base station block diagram

The Remote station was also set up similarily to phase 1 it with a bluetooth chip and missing the shield, joystick and light sensor. Each of the coloured wires in Figure 3 corresponding to the following inputs and outputs:

- 🔴 Red - laser component
- 🔵 Blue - Bluetooth chip
- 🟤 Brown - Tilt servo
- 🟠 Orange 2 - Pan servo
- 🟡 Yellow - Ground
- 🟢 Green - VCC

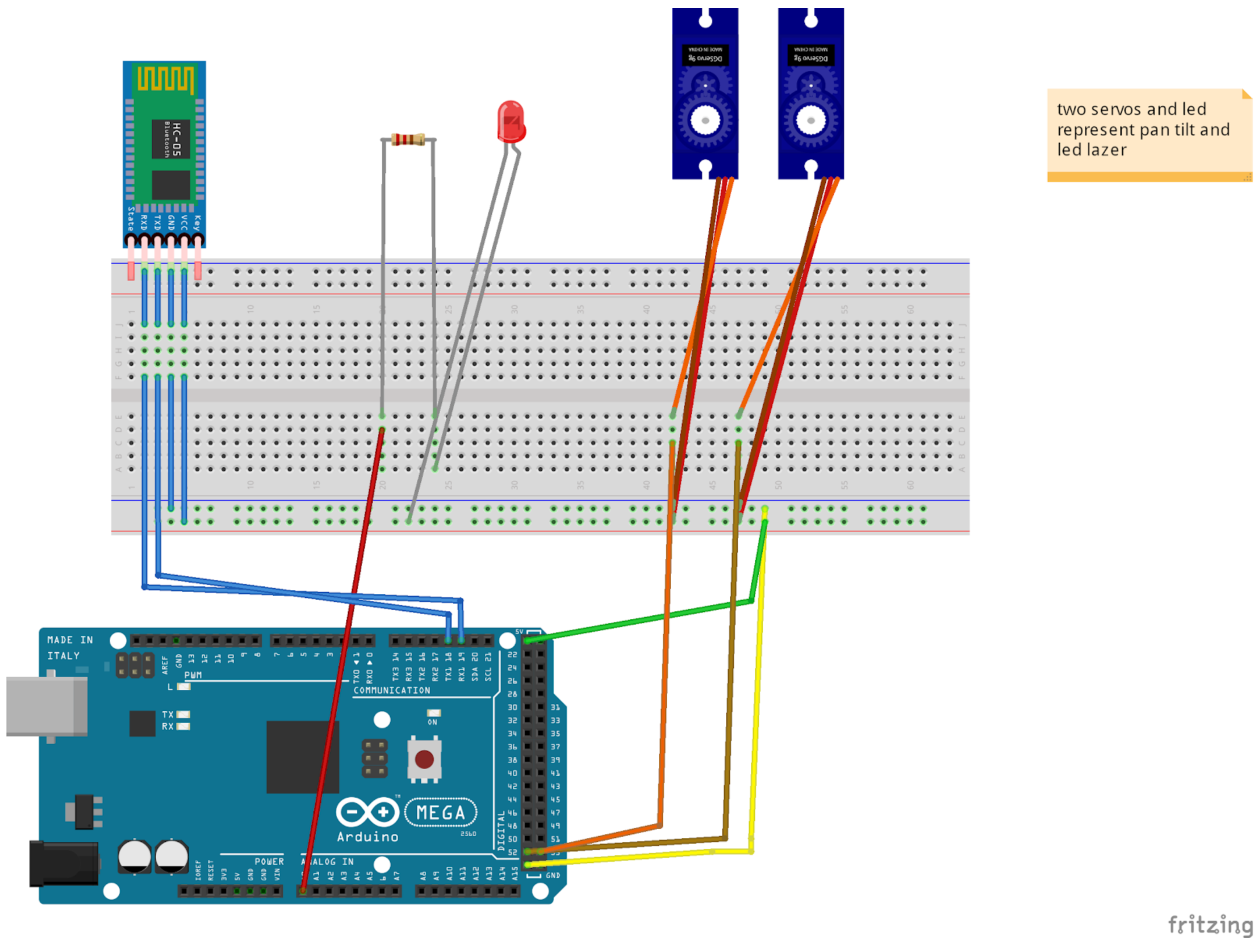

Figure 3 : Phase 2 Remote station block diagram

Bluetooth

In phase 2, communication over bluetooth was introduced, with one board taking user input and the other controlling the servo motors and lasers. The base station arduino board was taking input (integers ranging from 0 to 1023) from the joystick (x-axis, y-axis, and push-button) and applying a low-pass filter. These input values are then used to calculate the desired speed of the servo motors, which is then mapped to a range, between 0 and 1023, while ignoring the deadband in the centre of the ranges. This range was was then mapped to a smaller range of integers: 3-84 in the case of the x-axis and 1-22 in the case of the y-axis, 1|2 in the case of the push-button. These integers were used to send data over bluetooth, two bytes at a time, with the first byte specifying the type of input (1:laser, 1:x-axis, 2:y-axis) and the second byte being the mapped value mentioned.

Since two bytes were sent to indicate what input or range needed to be sampled, at some point the Base and Remote stations would become out of sync, sending incorrect bytes to the remote station which would result in glitchy, repetitive behaviour. In order to stop this from occurring byte 0 was reserved as a reset bit, sent periodically to clear the bluetooth buffer. This ensures that in case the receiving bluetooth chip gets out of sync, the system can correct itself.
Another problem that was encountered was that since the speed was originally being filtered on the base station when latency was introduced or there were some discrepancies with the received bytes the speed adjustments would get skipped. Because of this the relative position of the pan and tilt pins would start to jitter or in some cases jump erratically. This would happen due to reset bits or lost bytes and it was decided that the smoothing of the speed ensuring that the difference was no more than 2 different from the previous recorded speed should be performed on the remote station.

TTS: Time-triggered Scheduler

Another improvement that was made was due to the polling of the joystick controls, light sensor checks, sending data over bluetooth, and all outputs to the LCD where all happening in a single loop function, with a delay of 50ms. In order to plan the execution of each of these activities, a time-triggered scheduler was implemented [3] and a task was made for each activity, with the scheduler executing these tasks at a period specified during setup. In addition to these tasks, the actions on the remote station of controlling the laser and servo motors, as well as reading the data being sent over bluetooth were changed to be controlled by a time-triggered scheduler, and the maximum periods for those tasks were also found. The periods that were chosen as the maximum period for each task that would still result in an overall responsive system. The findings for these maximum periods can be seen in the table below.

An issue that occurred while switching over to this time-triggered scheduling approach was the reset byte that was sent in order to ensure the base and remote station were properly synchronised was being sent too often, as it was initially placed in the idle task, which the TTS would call whenever there was idle time (no other tasks needed to run). Given the low CPU utilization due to the relatively large periods chosen and short runtime of each task, this idle task happened so often that the resulting amount of reset bytes sent overwhelmed the remote station causing erratic behaviour and delays. This was solved by creating a task for sending a single reset byte with a period of 2 seconds. This meant that the two stations could not be out of sync for more than 2 seconds, but the reset byte would not be sent too often.

In order to check the CPU utilization of the tasks performed by the base and remote stations, high and low bits were set to pins on the start and completion of tasks. These were connected from both stations to a logic analyser, in order to measure the frequency, length of each task and overall period of tasks on each station. These values would be used to calculate the CPU utilization and adjusted to find the longest possible periods to keep the system working smoothly. Since the logic analyser was connected to both arduino boards initially, the ground connection had only been connected to one arduino board, the one that had more tasks, the Base station. However, because of this the output for the two tasks and the Idle task were erratic and all the readings were the same. When the logic analyser ground was connected to the ground of the Remote station as well, the periods became consistent with the rates set in code and the CPU utilization could be calculated. Because of this, it is likely only possible to read pins into the logic analyser from two boards at a time, as it only has two ground pins.

The logic analyzer was used to observe the occurrences and runtimes and a capture of this data can be seen below in figure 4 and figure 5.

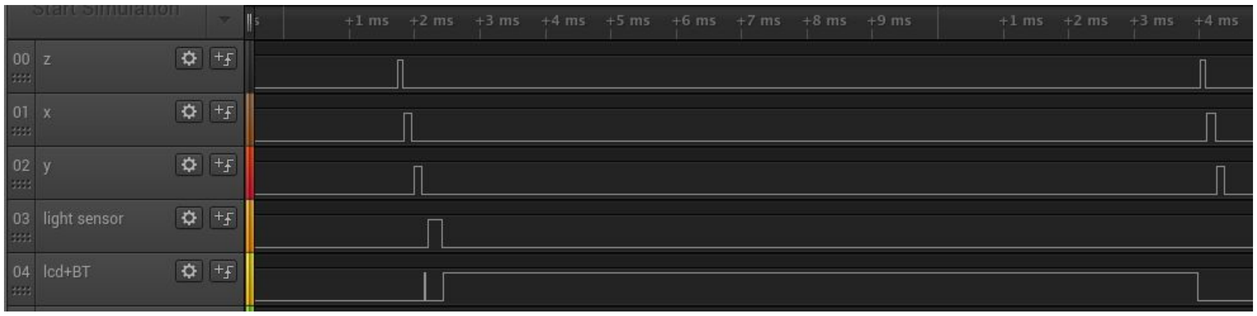

Figure 4: runtimes and occurrences of base station tasks

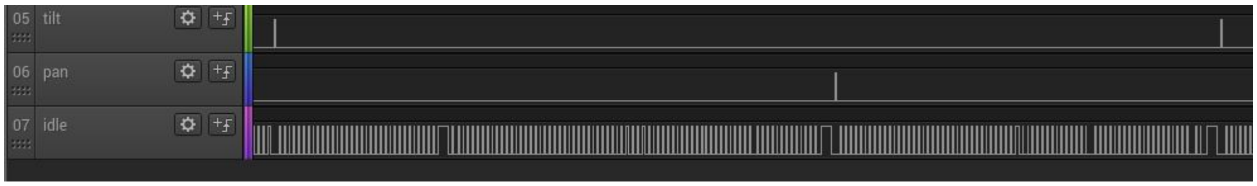

Figure 5: runtimes and occurrences of remote station tasks

Base station

Task	Width/Runtime (ms)	Occurrences	Total Runtime
Poll X-axis joystick input	0.12	12	1.44
Poll Y-axis joystick input	0.12	12	1.44
Poll joystick push-button input	0.065	12	0.78
Light sensor	0.21	1	0.21
LCD	11.53	1	11.53
Send bluetooth data	0.027	5	0.135
Reset byte over bluetooth	0.0003125	1	0.0003125

Overall total runtime = 15.535ms
Overall period = 1205.984ms
CPU Utilization = (Overall total runtime / Overall period) / 100 = 1.29 %
Longest period with which the system as a whole still performs smoothly:

- Poll X-axis joystick control input: Tmax = 10ms
- Poll Y-axis joystick control input: Tmax = 10ms
- Poll Z-axis joystick push button input: Tmax = 10ms
- Light sensor task = 120ms
- LCD task = 120ms
- Send bluetooth = 20ms
- Reset bluetooth = 2 sec

Remote station

Task	Width/Runtime (ms)
Poll tilt servo output	0.0063
Poll pan servo output	0.0181

Overall total runtime = 0.0181 ms
Overall period = 19.46 ms
CPU Utilization = (Overall total runtime / Overall period) / 100 = 0.095%

Longest period with which the system as a whole still performs smoothly:

- Tilt task T max = 20ms
- Pan task T max = 20ms

4.0 Conclusion

The aim of the project was to familiarise with Arduino hardware and software interfacing. With the help of the logic analysers, online sample code and Arduino code examples, among others, the project could be realised. It was helpful to print the relevant information using the Serial.print functionality and using a logic analyser to time the tasks and see any patterns in the signals being sent. Most of the difficulties arose when circuits partially disconnected, which was particularly confusing when it occurred after some functionality was successfully implemented and, due to disconnections, the system no longer performing in the same way. Other difficulties arose when trying to send data over bluetooth or implement the time-triggered schedule for each task. Difficulties also arose due to bluetooth connection, latency or sampling rate issues. For example, whenever the remote station *read bluetooth* task would be placed in a scheduled event rather than the idle loop it would increase latency, as the bluetooth information wasn't immediately read and updated.

5.0 References

[1] LiquidCrystal Library. Available online at: https://www.arduino.cc/en/Reference/LiquidCrystal
[2] Photocell Hookup Guide. Available online at:
https://learn.sparkfun.com/tutorials/photocell-hookup-guide/all
[3] Lab guide: Time-triggered scheduling. Available online at:
https://nrqm.ca/mechatronics-lab-guide/lab-guide-time-triggered-scheduling/
[4] Project 1 Code: Phase 1 and 2. Available online at:
https://github.com/HamzahMansour/csc460

Appendix A

phase1.ino

```
#include <Servo.h>
#include <LiquidCrystal.h>

// select the pins used on the LCD panel
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
int lcd_key     = 0;
int adc_key_in  = 0;

double z1 = 1;
double z0;
int x1 = 514;
int y1 = 506;
int z, x0,x, y0, y;
int JS1Z = 22;
int JS1X = A8;
int JS1Y = A10;
int PAN = 53;
int TILT = 52;
int LZ = A7;
//tiltvars
int maxAdjustTilt = 10;
int tiltSpeed = 0;
static unsigned int tiltValue = 500;
//panvars
int maxAdjustPan = 40;
int panSpeed = 0;
static unsigned int panValue = 2100;
Servo panServo;
Servo tiltServo;

// Light sensor and LED
const int LIGHT_PIN = A15; // Pin connected to voltage divider output
const int LED_PIN = 26; // Use built-in LED as dark indicator
// Measure the voltage at 5V and the actual resistance of your
// 47k resistor, and enter them below:
const float VCC = 4.98; // Measured voltage of Ardunio 5V line
const float R_DIV = 4660.0; // Measured resistance of 3.3k resistor
// Set this to the minimum resistance require to turn an LED on:
const float DARK_THRESHOLD = 130000.0;

void setup() {
  cli();
  pinMode(JS1Z, INPUT_PULLUP);
  pinMode(JS1X, INPUT);
  pinMode(JS1Y, INPUT);
  pinMode(LZ, OUTPUT);
  Serial.begin(9600);
  panServo.attach(PAN, 750, 2100);
  tiltServo.attach(TILT, 500, 1000);
  // timer setup for tilt
  TCCR4A = 0;
  TCCR4B = 0;
  //Set prescaller to 256
  TCCR4B |= (1<<CS42);

  //Set TOP value (0.5 seconds)
  OCR4B = 1736;

  //Enable interupt A for timer 3.
  TIMSK4 |= (1<<OCIE4B);

  TCNT4 = 0;

  // timer setup for pan
  // clear timer config
  TCCR3A = 0;
  TCCR3B = 0;
  //Set prescaller to 256
  TCCR3B |= (1<<CS32);

  //Set TOP value (0.5 seconds)
  OCR3B = 1736;

  //Enable interupt A for timer 3.
  TIMSK3 |= (1<<OCIE3B);

  //Set timer to 0 (optional here).
  TCNT3 = 0;
  sei();

  // LCD things
  lcd.begin(16, 2);              // start the library
  lcd.setCursor(0,0);
  lcd.print("LGHT  JSX  JSY Z");

  // Light sensor and LED
  pinMode(LIGHT_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);

}
// tilt handling
```

```
/*
At this point tilt doesnt appear to be working I'm going to head home also
there was some inexplicable shange in our x axis range
*/
ISR(TIMER4_COMPB_vect){
  int tiltInput = y1;
  int oldSpeed = tiltSpeed;
  if(tiltInput > 570){
    tiltSpeed = map(tiltInput, 570, 1021, 0, -maxAdjustTilt);
  }
  else if(tiltInput < 440){
    tiltSpeed = map(tiltInput, 440, 0, 0, maxAdjustTilt);
  }
  else{
    tiltSpeed = 0;
  }
  if(oldSpeed - tiltSpeed > 2){
    tiltSpeed = oldSpeed - 2;
  }
  else if( oldSpeed - tiltSpeed < -2){
    tiltSpeed = oldSpeed + 2;
  }
  tiltValue += tiltSpeed;
  if(tiltValue > 1000)
    tiltValue = 1000;
  if(tiltValue < 500)
    tiltValue = 500;

  if(tiltSpeed != 0)
    tiltServo.writeMicroseconds(tiltValue);
  TCNT4 = 0;
}
//pan handling
ISR(TIMER3_COMPB_vect){
  int panInput = x1;
  int oldSpeed = panSpeed;
  if(panInput > 575){
    panSpeed = map(panInput, 525, 1022, 0, -maxAdjustPan);
  }
  else if (panInput < 425){
    panSpeed = map(panInput, 425, 0, 0, maxAdjustPan);
  }
  else{
    panSpeed = 0;
  }

  if(oldSpeed - panSpeed > 2){
    panSpeed = oldSpeed - 2;
  }
  else if( oldSpeed - panSpeed < -2){
    panSpeed = oldSpeed + 2;
  }
  panValue += panSpeed;

  if(panValue > 2100)
    panValue = 2100;
  if(panValue < 750)
    panValue = 750;

  if (panSpeed != 0)
    panServo.writeMicroseconds(panValue);
  TCNT3 = 0;
}

void loop() {
  // lazer handling
  z0 = z1;
  z = digitalRead(JS1Z);
  z1 = z0 + ((z-z0)*0.3);
  z = (int)(z1 + 0.5);
  analogWrite(LZ, (int)(1-z)*200);
  // pan handling
  x0 = x1;
  x = analogRead(JS1X);
  x1 = x0 + ((x-x0)>>1);
  //tilt handling
  y0 = y1;
  y = analogRead(JS1Y);
  y1 = y0 + ((y-y0)>>1);

  // LCD print joystick X,Y,Z
  lcd.setCursor(5,1);
  lcd.print("           ");//0000 0000 0");
  lcd.setCursor(5,1);
  char printfStr[16];
  sprintf(printfStr,"%4d %4d %d",x1, y1,(int)(z1 + 0.5));
  lcd.print(printfStr);

  // Light sensor and LED
  // Read the ADC, and calculate voltage and resistance from it
  int lightADC = analogRead(LIGHT_PIN);
  if (lightADC > 0)
  {
    // Use the ADC reading to calculate voltage and resistance
    float lightV = lightADC * VCC / 1023.0;
    float lightR = R_DIV * (VCC / lightV - 1.0);
    // If resistance of photocell is greater than the dark
    // threshold setting, turn the LED on.
    if (lightR >= DARK_THRESHOLD){
      digitalWrite(LED_PIN, LOW);
      lcd.setCursor(0,1);
      lcd.print("SAFE");
    }
    else{
      digitalWrite(LED_PIN, HIGH);
      lcd.setCursor(0,1);
      lcd.print("SHOT");
    }
  }

}
```

Phase2
Base.ino

```
#include "Arduino.h"
#include "scheduler.h"
#include <LiquidCrystal.h>

// select the pins used on the LCD panel
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
int lcd_key  = 0;
int adc_key_in  = 0;

int x, y, z, x0, y0, x1=506, y1=514, z_prev=1;
double z0, z1=1;
int JS1X=A8, JS1Y=A9, JS1Z=22;
int panSpeed = 0, maxAdjustPan = 40;
int tiltSpeed = 0, maxAdjustTilt = 10;
int panByte=3, tiltByte=1;
boolean shot=false;

const int LIGHT_PIN = A15;
const float VCC = 4.98;
const float R_DIV = 4660.0;
const float DARK_THRESHOLD = 100000.0;

// task function for joystick z-axis input pin (polling)
void poll_z_task(){ // every 10 ms
  digitalWrite(30, HIGH);
  // laser handling
  z0 = z1;
  z = digitalRead(JS1Z);
  z1 = z0 + ((z-z0)*0.3);
  z = (int)(z1 + 0.5);
  if(z != z_prev){
      z_prev = z;
      if(z==1) writeTwo(1,1);
      else if(z==0) writeTwo(1,2);
  }
  digitalWrite(30, LOW);
}

void poll_x_task(){ // every 20ms
  digitalWrite(31, HIGH);
  // pan handling (x: 1,3 -> 1,44)
  x0 = x1;
  x = analogRead(JS1X);
  x1 = x0 + ((x-x0)>>1);
  panHandle(x1);
  digitalWrite(31, LOW);
}

void poll_y_task(){ // every 20ms
  digitalWrite(32, HIGH);
  //tilt handling
  y0 = y1;
  y = analogRead(JS1Y);
  y1 = y0 + ((y-y0)>>1);
  tiltHandle(y1);
  digitalWrite(32, LOW);
}

void light_sensor_task(){ // every 50ms
  digitalWrite(33, HIGH);
  int lightADC = analogRead(LIGHT_PIN);
  int a=0;
  if (lightADC > 0) {
      // Use the ADC reading to calculate voltage and resistance
      float lightV = lightADC * VCC / 1023.0;
      float lightR = R_DIV * (VCC / lightV - 1.0);
      // If resistance of photocell is greater than the dark
      if (lightR >= DARK_THRESHOLD){
      if(!shot) shot=true;
      }
      else{
      if(shot) shot=false;
      }
  }
  digitalWrite(33, LOW);
}

void lcd_display_task(){ // every 120ms
```

```
  digitalWrite(34, HIGH);
  // LCD print joystick X,Y,Z
  lcd.setCursor(0,1);
  lcd.print("             ");
  if(shot){
      lcd.setCursor(0,1);
      lcd.print("SAFE");
  } else {
      lcd.setCursor(0,1);
      lcd.print("SHOT");
  }
  lcd.setCursor(5,1);
  char printfStr[16];
  sprintf(printfStr,"%4d %4d %d",x1, y1,(int)(z1 + 0.5));
  lcd.print(printfStr);
  digitalWrite(34, LOW);
}

void send_bluetooth_task(){ // every 50ms
  digitalWrite(34, HIGH);
  writeTwo(1,panByte);
  writeTwo(2,tiltByte);
  digitalWrite(34, LOW);

}

void poll_reset_bluetooth(){ // every 2 sec
  digitalWrite(34, HIGH);
  Serial1.write(0);
  digitalWrite(34, LOW);
}

// idle task
void idle(uint32_t idle_period){
  // send bluetooth?
}

void setup(){
  Scheduler_Init();
  Scheduler_StartTask(0, 2, poll_z_task);
  Scheduler_StartTask(10, 5, poll_x_task);
  Scheduler_StartTask(20, 5, poll_y_task);
  Scheduler_StartTask(30, 5, send_bluetooth_task);
  Scheduler_StartTask(40, 120, light_sensor_task);
  Scheduler_StartTask(50, 120, lcd_display_task);
  Scheduler_StartTask(60, 2000, poll_reset_bluetooth);

  Serial.begin(9600);
  Serial1.begin(9600);
  pinMode(JS1Z, INPUT_PULLUP);
  pinMode(30, OUTPUT);
  pinMode(31, OUTPUT);
  pinMode(32, OUTPUT);
  pinMode(33, OUTPUT);
  pinMode(34, OUTPUT);
  for(int i=30; i<35; i++) digitalWrite(i, LOW);

  // LCD things
  lcd.begin(16, 2);   // start the library
  lcd.setCursor(0,0);
  lcd.print("LGHT  JSX  JSY Z");

  // Light sensor
  pinMode(LIGHT_PIN, INPUT);
}

void writeTwo(int a, int b){
  Serial1.write(a);
  Serial1.write(b);
}

void panHandle(int panInput){
  int oldSpeed = panSpeed;
  if(panInput > 540){
   panSpeed = map(panInput, 540, 1021, 0, -maxAdjustPan);
  }
  else if (panInput < 440){
   panSpeed = map(panInput, 440, 0, 0, maxAdjustPan);
  }
  else{
   panSpeed = 0;
  }

  if(panSpeed != oldSpeed){
   // set a global variable or array to a value
   panByte = map(panSpeed, -maxAdjustPan, maxAdjustPan-1, 3, 84);
  }
}

void tiltHandle(int tiltInput){
  int oldSpeed = tiltSpeed;

  if(tiltInput > 555){
   tiltSpeed = map(tiltInput, 555, 1022, 0, maxAdjustTilt);
  }
  else if(tiltInput < 425){
   tiltSpeed = map(tiltInput, 425, 0, 0, -maxAdjustTilt);
  }
  else{
   tiltSpeed = 0;
  }

  if(tiltSpeed != oldSpeed){
   // set a global variable or array to a value
   tiltByte = map(tiltSpeed, -maxAdjustTilt, maxAdjustTilt-1, 1, 22);
  }
}

void loop(){
  uint32_t idle_period = Scheduler_Dispatch();
  if(idle_period){
   idle(idle_period);
  }
}

int main(){
    init();
    setup();
    for (;;){ loop(); }
    for (;;);
    return 0;
}
```

Remote.iso

```
#include "scheduler.h"
#include <Servo.h>
int command[2] = {0,0};
int indx = 0;
int LZ = A15, PAN = 53, TILT = 52;
int panValue = 2050;
int tiltValue = 950;
int maxAdjustTilt = 10;
int maxAdjustPan = 40;
int speedTilt = 0;
int oldTiltSpeed = 0;
int speedPan = 0;
int oldPanSpeed = 0;
int panResponce = 0;
int tiltResponce = 0;
int readPin = 32,tiltPin = 30,panPin = 31;
Servo panServo;
Servo tiltServo;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial1.begin(9600);
  pinMode(LZ,OUTPUT);
  panServo.attach(PAN);
  tiltServo.attach(TILT);
  pinMode(readPin, OUTPUT);
  digitalWrite(readPin, LOW);
  pinMode(tiltPin, OUTPUT);
  digitalWrite(tiltPin, LOW);
  pinMode(panPin, OUTPUT);
  digitalWrite(panPin, LOW);
  Scheduler_Init();

  // Start task arguments are:
  //    start offset in ms, period in ms, function callback

  Scheduler_StartTask(0, 5, tiltHandler);
  Scheduler_StartTask(2, 5, panHandler);
  analogWrite(LZ,0);
}
// handling tilt
void tiltHandler(){
  digitalWrite(tiltPin, HIGH);
  int tempT = speedTilt;
  if(oldTiltSpeed > tempT){
    tempT = oldTiltSpeed - 1;
  }
  else if(oldTiltSpeed < tempT){
    tempT = oldTiltSpeed + 1;
  }
  if(tiltValue + tempT > 1000 || tiltValue + tempT < 500){
    tempT = 0;
  }

  oldTiltSpeed = tempT;
  if(tempT != 0) {
    tiltValue += tempT;
    if(tiltValue >= 1000)
      tiltValue = 1000;
    if(tiltValue <= 500)
      tiltValue = 500;
    tiltServo.writeMicroseconds(tiltValue);
  }
  digitalWrite(tiltPin, LOW);

}
//pan handling
```

```
void panHandler(){
  digitalWrite(panPin, HIGH);
  int tempP = speedPan;
  if(oldPanSpeed - tempP > 2){
    tempP = oldPanSpeed - 2;
  }
  else if( oldPanSpeed - tempP < -2){
    tempP = oldPanSpeed + 2;
  }
  if(panValue + tempP > 2100 || panValue + tempP < 750){
    tempP = 0;
  }
  oldPanSpeed = tempP;
  if(tempP != 0) {
    panValue += tempP;
    if(panValue >= 2100)
      panValue = 2100;
    if(panValue <= 750)
      panValue = 750;
    panServo.writeMicroseconds(panValue);
  }
  digitalWrite(panPin, LOW);

}

void callFunction(){
  int v0 = command[0],v1 = command[1];
  switch(v0){
    case 1:
      switch(v1){
        case 1:
            analogWrite(LZ,200);
            break;
          case 2:
            analogWrite(LZ,0);
            break;
          case 3 ... 84:
            speedPan = map(v1, 3, 84, -maxAdjustPan, maxAdjustPan);
            break;
          default: return;
        }
        break;
       case 2:
       switch(v1){
       case 1 ... 22:
            speedTilt = map(v1, 1, 22, -maxAdjustTilt, maxAdjustTilt);
            break;
            default: return;
```

```
     }
     break;
     default: return;

  }
   //Serial.write(v0);
   //Serial.write(v1);
}

void readSerial1(){
  digitalWrite(readPin, HIGH);
  // put your main code here, to run repeatedly:
  if(Serial1.available()){
    int readVal = Serial1.read();
    if(readVal == 0) {
      indx = 0;
      digitalWrite(readPin, LOW);
      return;
    }
    command[indx++] = readVal;
    if(indx > 1){
      callFunction();
      indx = 0;
    }
  }
  digitalWrite(readPin, LOW);
}

// idle task
void idle(uint32_t idle_period)
{
  // this function can perform some low-priority task while the scheduler has nothing to run.
  // It should return before the idle period (measured in ms) has expired. For example, it
  // could sleep or respond to I/O.
  readSerial1();
}

void loop() {

  uint32_t idle_period = Scheduler_Dispatch();
  if (idle_period)
  {
    idle(idle_period);
  }
}

```

```c
int main()
{
  init();
  setup();

  for (;;)
  {
    loop();
  }
  for (;;);
  return 0;
}
```