



RAILWAY CROSSING SIMULATION

2017 Year 12 Systems Engineering Project (SAT)

By Berk Dogan



19-Sep-2017
SYSTEMS ENGINEERING SAT

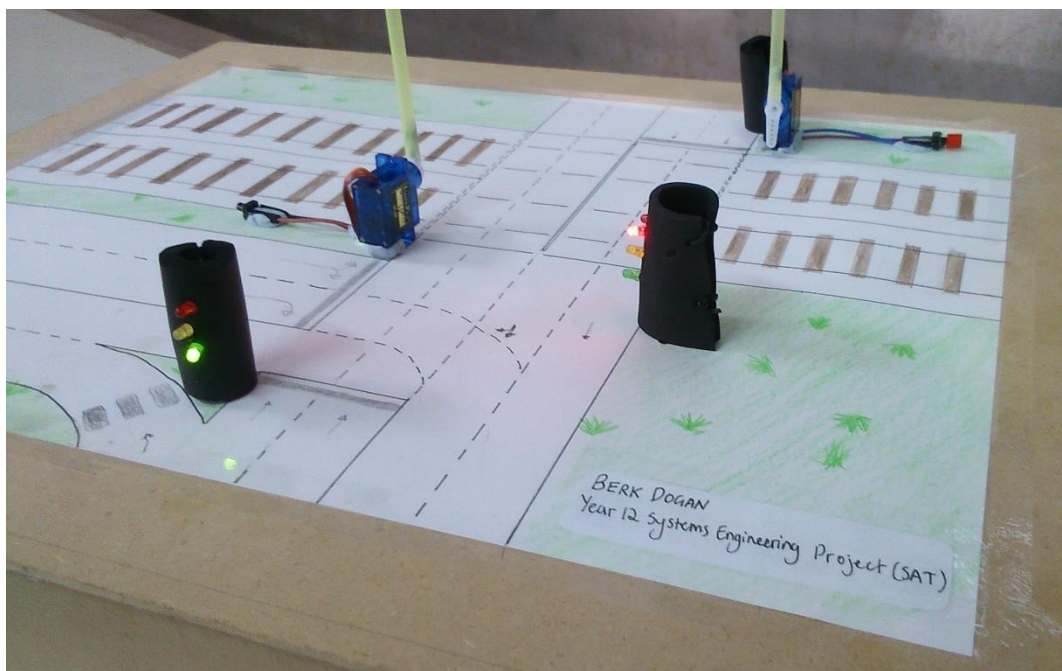
Table of Contents

Introduction	2
Design Plan	4
Design Brief	4
Design Options	6
Considerations	12
Production Plan	13
Time Line Chart	13
Components for Option 3	15
Risk Assessment	18
Project Build	19
Model Construction	19
Issues (Materials)	21
Electronics	22
Issues (Electronics)	24
Schematic	25
Programming	26
Issues (Programming)	33
Testing	34
Test #1	34
Test #2	34
Test #3	35
Test #4	37
Evaluation	38
Planning	38
Improvements	38
Evaluation of Components	39

Introduction

This project aims to produce a working model of a railway crossing together with a side street.

As represented below we have a North-South road that crosses the railway line which is on an East-West design. On the North side of the railway line is an intersecting East-road. The East-road ends at the intersection of the North-South Road. The image below is a bit deceiving because we used the top of the image to represent the south and the bottom to represent the North. The side street is on the north of the rails and is a west approaching to the intersection.



The North-South road takes precedence when the system is activated.

When a train is sensed, the software program sets into motion and stops the flow of all active traffic on the North-South. The lights go red and the boom gates come down after a few seconds. We've given about 30 seconds to represent a train go

through the intersection. After the about 30 second interval (we can alter this to suite our needs) the boom gate goes back up and then the North-South road has green lights.

In the event that a car is detected on the East road then another routine is executed. There are 2 options here that is possible. The North-South traffic must have at least be given the allotted time (in our case 30 seconds) to full-fill. If 30 seconds has already been given to the North-South then the routine to stop the North-South traffic is executed immediately otherwise we have to wait for 30 seconds for the condition North-South traffic to be fulfilled.

In another event that a train arrives during the East-road then the traffic lights for left turns must be set to red but the right turning traffic continues to get a green.

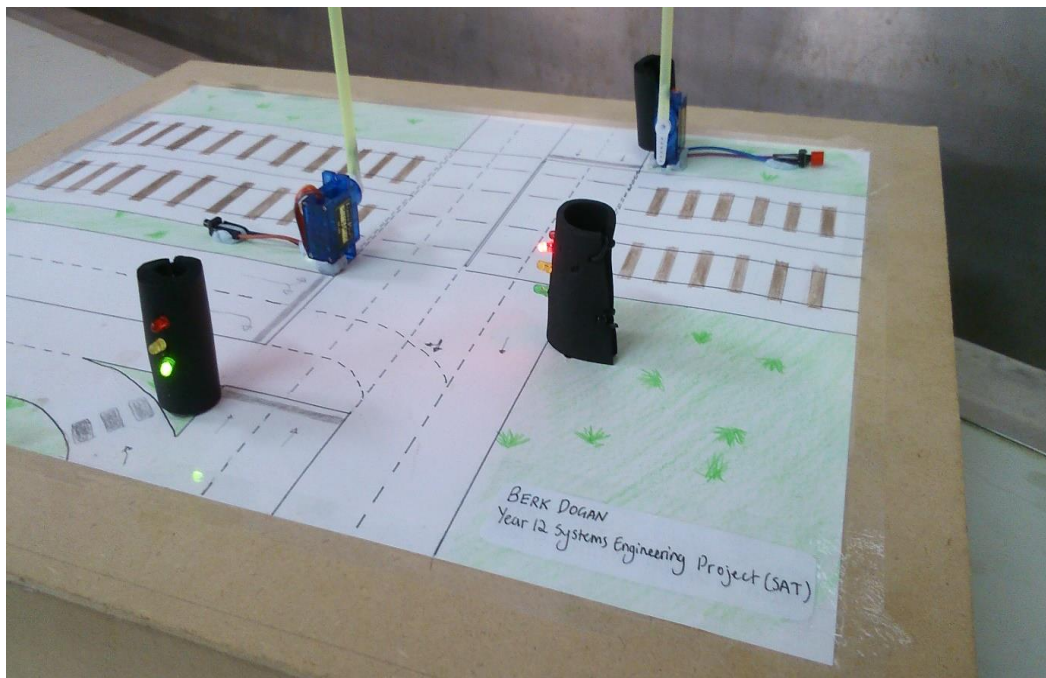
DESIGN PLAN

DESIGN BRIEF

This is my Year 12 Systems Engineering School Assessment Task (SAT). I chose to create a **Railway Crossing Simulation** which is designed to show how an intersection with incoming roads interact with the level crossing.

The research required me to first understand how all the various roads and the rail would interact together with a control system implement to monitor and set traffic lighting conditions.

After researching the level crossing and considering various construction methods for the railway crossing simulation, the project research and development resulted in three design options where two of the three are roughly the same however, one design required the use of an ultrasonic sensor to detect activity in any one train line or road.



To implement this project the required technical skills and knowledge were required in mathematics, electronics and computer programming. Finally, a design was selected out of the 3-initial researched.

The first design that was used ended up very simplistic and required little to no programming knowledge, I later abandoned that project idea. **Design Option 2** was created which was a huge improvement over the first design however, I had to learn a lot about the **Arduino programming language**. Design option 2 required ultrasonic sensors but due to faulty sensors that design option was revised and ended up with the **3rd and final design option**. The final design used switches in favour of ultrasonic sensors.

Regardless of which design option we needed go with the project required some effort to create a model of the railway crossing simulation onto a board for visually displaying the conditions that would occur in real life.

This project would be a government project if it were used in a real scenario. A crossing like this would be a result of new roads and rails coming together or an upgrade to an existing controlling system. Although this is a simulation at a small scale, it can be replicated for real world use. The programming and routines in my railway crossing simulation has been made to detect switches instead of the sensors that a real intersection may have like magnetic and ultrasonic sensors.

Due to the limitations of the Arduino board, there were only 2 sensor connections that was available for use and that was enough to simulate the conditions of traffic and train arriving from either direction.

DESIGN OPTIONS

As mentioned in the design brief, there were three design options to choose from to have a completed project. As it turned out option 3 was the one I took on.

OPTION 1

Option 1 was initially going to be a recreation of the Yarraman crossing but making a few changes to the layout. At first, I was planning on using LEGOs to demonstrate the physical model of the Railway Crossing. However, things didn't go the way I planned.

Using colourful blocks would have made the overall appearance of the project childish and the layout wouldn't have been original. There was going to be an underpass for a car lane but the LEGOs turned out to be a limiting factor. In the end, I didn't go with design option 1.

Design option 1 materials, tools and costs as outlined below.

MATERIALS (Circuit)

- Arduino UNO R3
- Male to male jumper cables
- 330R resistors
- 10K resistors
- Assorted coloured wires
- Red LEDs (uses 330R)
- Green LEDs (uses 330R)
- Yellow LEDs (uses 330R)
- SG90 servos
- 2-pin switches
- USB data cable
- Breadboard

MATERIALS (Build)

- 400 x 350mm MDF (not in good shape)
- Coloured Paper
- Blu Tack
- Straws for boom-gates
- LEGOs ("Perfect-Lock" bricks) in assorted colours
- Popsicle sticks

TOOLS

- Scissors
- Blu-stick
- Electrical Tape
- Soldering Iron + Solder

COST - \$70

OPTION 2

Option 2 and option 3 are very similar but they are completely different from option 1 as I have redesigned the railway crossing and intersections and no longer mimic the Yarraman crossing.

There are new variables such as “no left-turn” and the use of ultrasonic sensors. The ultrasonic sensors act as switches to indicate a train is coming and therefore it would initiate a certain sequence.

Although option 2 was going to be my actual design, I later realised that I had a damaged ultrasonic sensor. I tried to troubleshoot it however it appears that the ultrasonic sensor was damaged from the factory. Thus I had to compromise and use basic switches in its place to indicate a train is approaching the intersection. Thus, option 2 was abandoned in favour of option 3 design.

MATERIALS (Circuit)

- 1x Arduino UNO R3
- Male to male jumper cables
- 10x 330R resistors
- Assorted coloured wires (I made sure to use black for negative polarity to avoid confusion)
- 4x Red LEDs (uses 330R)
- 3x Green LEDs (uses 330R)
- 3x Yellow LEDs (uses 330R)
- 2x SG90 Servos
- 3x 2-pin switches
- 2x HR-04 Ultrasonic Sensors
- 1x USB data cable
- 1x 9V battery pack to power Arduino
- 1x Breadboard

MATERIALS (Build)

- 500 x 600mm MDF
- A3 Papers
- Straws for boom-gates
- LEGOs (“Perfect-Lock” bricks) in assorted colours
- Popsicle sticks

TOOLS Used

- Jig-saw
- Utility knife
- Assorted Colouring Pencils
- Drill
- Ruler/Tape measure
- Fine-liner
- Soldering Iron + Solder
- Electrical Tape
- Blu Tack

COST

Approximately \$100

(note: the tools used are not included as part of the overall cost and my components were at a different cost.) (refer to 2.4)

OPTION 3 (Selected)

Option 3 is essentially option 2 however without the ultrasonic sensors. As mentioned previously, unfortunately my ultrasonic sensor was not functioning correctly and thus went ahead and use alternative method of sensing the trains.

The pseudo-code was greatly improved on as well and I settled on just two switches as opposed to three. The overall complexity remained the same and the application became much more functional and as expected.

MATERIALS (Circuit)

- 1x Arduino UNO R3
- Male to male jumper cables
- 10x 330R resistors
- Assorted coloured wires (I made sure to use black for negative polarity to avoid confusion)
- 4x Red LEDs (uses 330R)
- 3x Green LEDs (uses 330R)
- 3x Yellow LEDs (uses 330R)
- 2x SG90 Servos
- 2x 2-pin switches
- 1x USB data cable
- 1x 9V battery pack to power Arduino
- 1x Breadboard

MATERIALS (Build)

- 500 x 600mm MDF
- A3 Papers
- Straws for boom-gates
- LEGOs ("Perfect-Lock" bricks) in assorted colours
- Popsicle sticks

TOOLS

- Jig-saw
- Utility knife
- Assorted Colouring Pencils
- Electric Drill
- Ruler/Tape measure
- Fine-liner
- Soldering Iron + Solder
- Electrical Tape
- Blu-Tack

COST

\$100

(note: does not include the cost of tools used and my components were at a different cost.)

CONSIDERATIONS

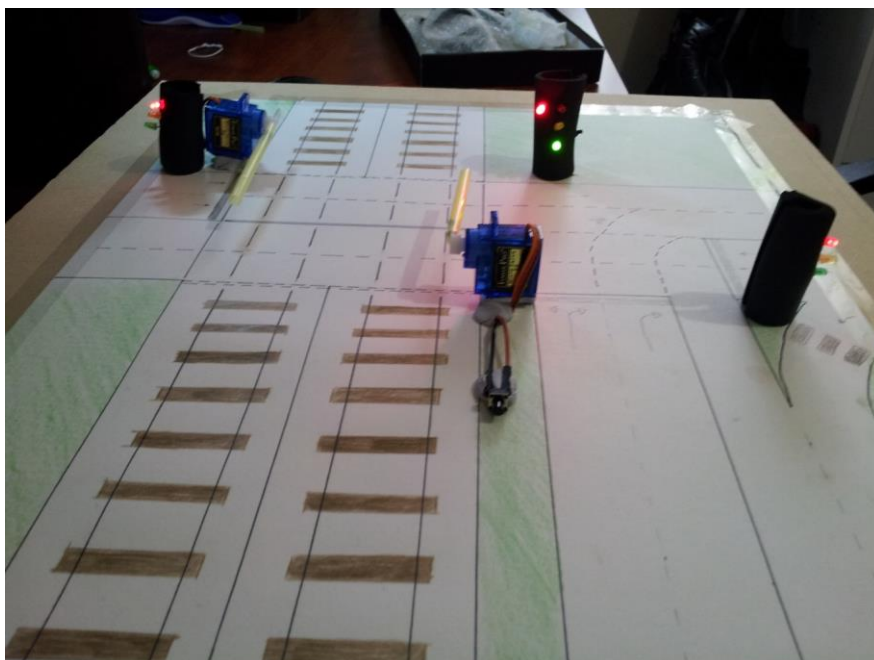
The railway crossing simulation requires one to have knowledge of the following component circuitry and programming:

- **Arduino circuitry** (specifically for the Arduino UNO R3)
- **Programming language** used the Arduino
- **Soldering** and manipulation of wires

This project involved research about and learning the Arduino circuit board. There was a learning curve to go through before I could understand how the Arduino components and the programming tied together to let me achieve my project tasks. Learning the Arduino Circuit Board and it's components along with the software needed to connect to it from a computer require many hours of research, experimentation and trial & error. Otherwise, I would not have completed to satisfaction.

Furthermore, I had to come up with a schematic on my own based on what I learned from the many sites used to understand the Arduino board and it's programming structure. The idea for the project and the work is all original work.

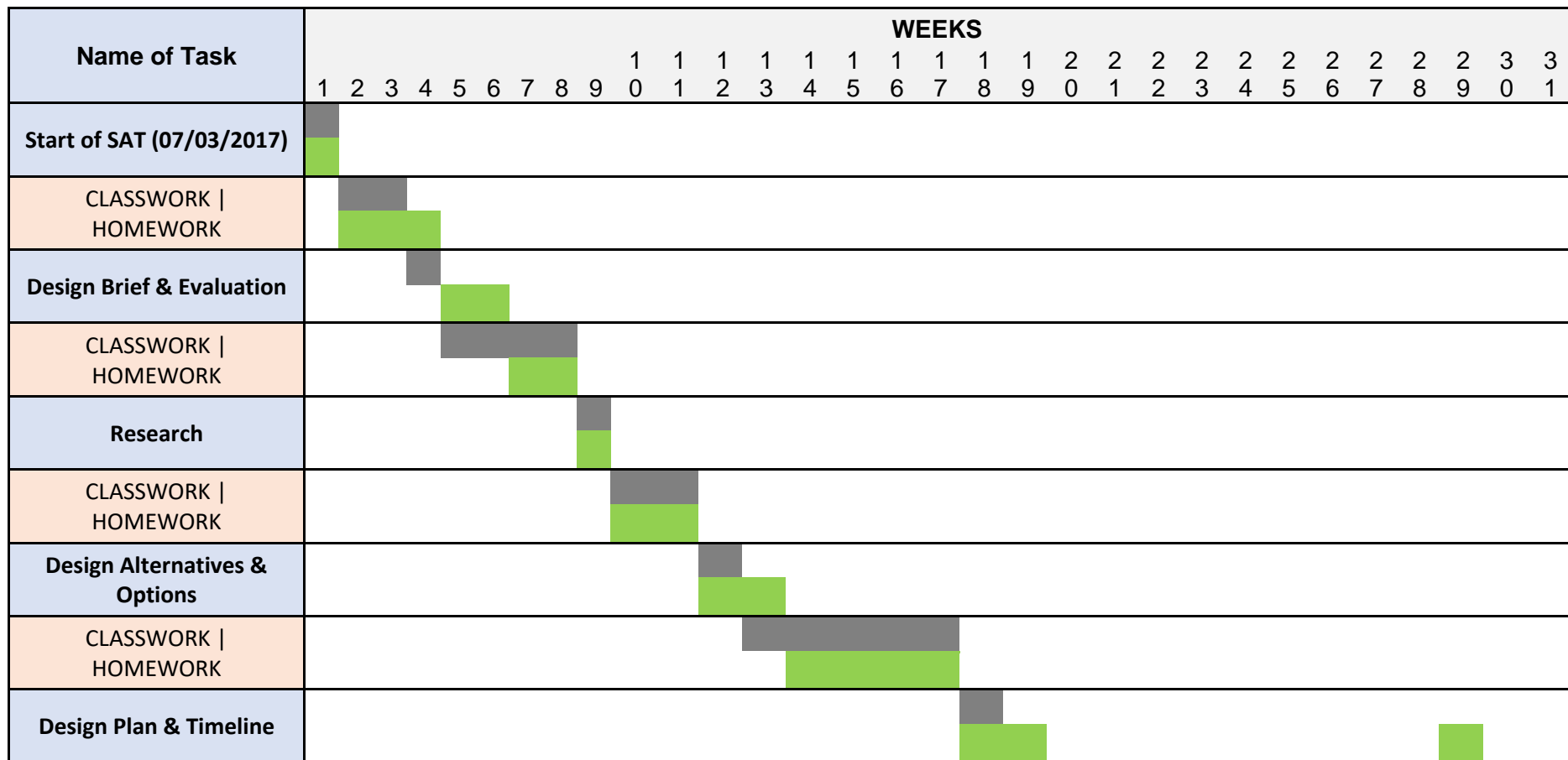
This is a 3D photo model of the railway crossing.



PRODUCTION PLAN

Time Line Chart

The grey boxes indicate the estimated time for completion and green is the actual time when it was finished.



COMPONENTS FOR OPTION 3

The table below is the complete parts list used to construct this project. The component prices as are sourced and with RRP.

Quantity	Product Description	Source	Cost
1	<p>Arduino Uno Revision 3</p> <p>The official Arduino Uno R3 can be purchased directly from Arduino's site. I didn't link any third-party boards that offer the same function as I wanted 100% compatibility.</p> <p>Note: Many of the boards made by third parties will still work.</p>	<p>Arduino (online)</p> <p>https://store.arduino.cc/arduino-uno-rev3</p>	\$29.87
1	<p>P1914B • 3m A Male to B Male USB 2.0 Patch Lead</p> <p>This cable is mandatory for uploading the sketch to the Arduino controller. Many kits come with these out of the box however consider purchasing one.</p> <p>Cables with the same end will work too, these cables generally come with printers and other devices.</p>	<p>Altronics (local)</p> <p>http://www.altronics.com.au/p/p1914b-3m-a-male-to-b-male-usb-2.0-patch-cable/</p>	\$7.50

1	R7040 • 330R 0.25W 5% Carbon Film Resistor PK 10 10 resistors rated at 330 ohms.	Altronics (local) http://www.altronics.com.au/p/r7040-330r-0.25w-carbon-film-resistor-pk-10/	\$0.50
1	Z0002 • 5mm LED Assortment Pack 100pcs A pack of 100 LEDs that come in green, red, yellow and orange colours. They will work well with 330R resistors.	Altronics (local) http://www.altronics.com.au/p/z0002-5mm-led-assortment-pack-100pcs/	\$17.50
2	S1122 • SPST Momentary PCB Mount 7mm Tactile Switch Two switches are required for the interrupt signals.	Altronics (local) http://www.altronics.com.au/p/s1122-spst-momentary-pcb-mount-7mm-tactile-switch/	\$0.95 (x1)
2	Z6392 • 9g 180° 2 x 90° Plastic Servo for Arduino Two servos are required for the boom-gates.	Altronics (local)	\$9.95 (x1)
1	P1002 • 400 Tie Point Interlocking Solderless Breadboard Although this would be enough for this project, it is advised that you purchase a bigger breadboard for easier resistor and jumper cable placement.	Altronics (local) http://www.altronics.com.au/p/p1002-400-way-breadboard/	\$12.50

1	P1016 • Male to Male Prototyping Wire Pack 65pcs These cables will be useful when connecting the LEDs and servos into the breadboard. 65pcs is good as we will be soldering some of them to electrical wire.	Altronics (local) http://www.altronics.com.au/p/p1016-prototyping-jumper-wires-male-to-male-for-breadboard-arduino/	\$5.95
N/A	Electrical Wire (insulated) (assorted colours) This can be purchased from many stores as they come in different lengths and colours. It is recommended that you get lots of black wire to help with identifying what is negative.	Most retailers	N/A
1	T3001A • 18mm Black Insulation Tape Electrical tape will be used to make sure the connections aren't shorting each other.	Altronics (Local) http://www.altronics.com.au/p/t3001a-18mm-black-insulation-tape/	\$2.40
1	3mm Standard MDF - 600mm x 900mm This board is a good size as it is more than enough for this project.	Bunnings (local) https://www.bunnings.com.au/3mm-standard-mdf-600mm-x-900mm_p0590066	\$2.33
Total: \$100.35 (excluding wire costs, tools and delivery costs)			

RISK ASSESSMENT

In this section, I analyse the risks associated with the use of various tools. These tools were used in the creation of the railway crossing simulation.

Hazard	Risk	Risk Control	PPE
Soldering Iron	<ul style="list-style-type: none"> Burns Flicking solder Burning components with excessive heat Fumes 	<ul style="list-style-type: none"> Should be used with care. Keep hot parts of the iron away from fingertips Avoid breathing in toxic fumes 	<ul style="list-style-type: none"> Safety goggles Safety mask
Jig-saw	<ul style="list-style-type: none"> Sharp blade Dangerous to operate Cutting wood Sound Electricity Air (saw-dust) 	<ul style="list-style-type: none"> Should only be operated in a clear area with no one around Avoid echoed rooms Keep electrical cord at a safe distance, away from blade. Be sure to be in a well-ventilated room or use a safety mask. 	<ul style="list-style-type: none"> Ear muffs Safety goggles Safety mask Blade guard
Hot glue gun	<ul style="list-style-type: none"> Heat (Burns) Electricity 	<ul style="list-style-type: none"> The electrical cable should be placed away from the heating element as it may be damaged by the heat. Avoid touching fresh hot-glue or the barrel as it may burn the user. 	<ul style="list-style-type: none"> Nothing Gloves (optional)
Drill	<ul style="list-style-type: none"> Loud Sharp 	<ul style="list-style-type: none"> Should be used with caution and used carefully. 	<ul style="list-style-type: none"> Ear muffs

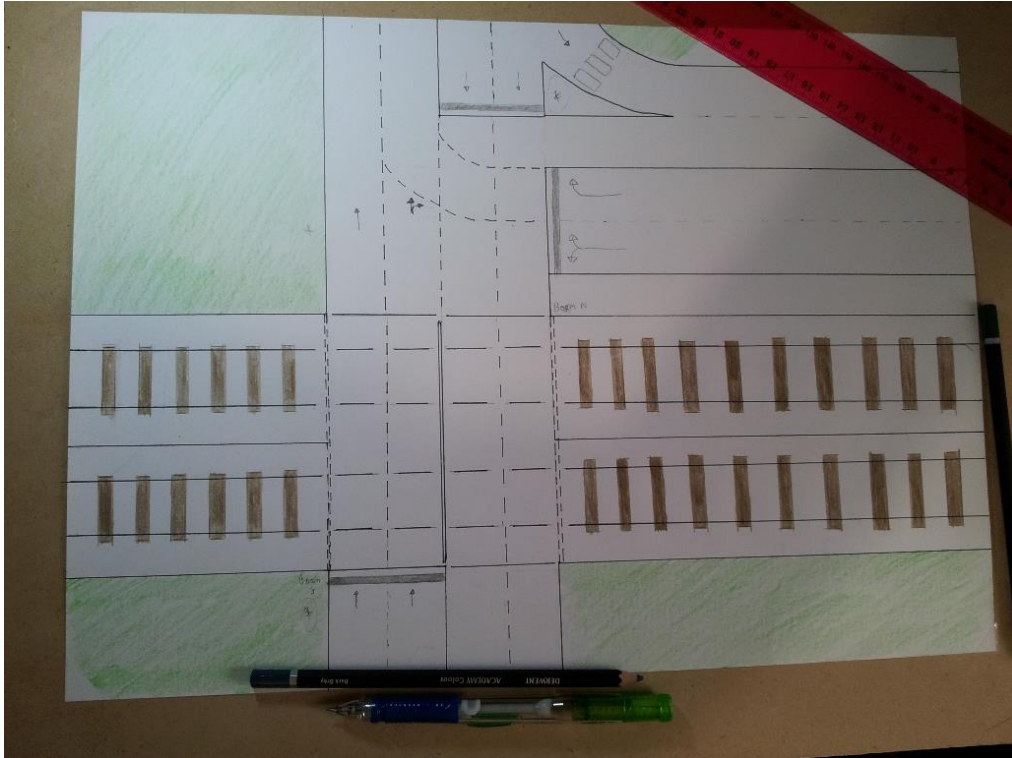
PROJECT BUILD

MODEL CONSTRUCTION

For this project, I wanted the simulation to be portable but also straight-forward to work on. As for the model box which the railway crossing simulation would be built on: MDF was the best suited material to use as this was easy to cut and place circuitry, sensors, lighting and other apparatus to represent the crossing. Furthermore, MDF made it easy to transport because it was light weight and portable.

The MDF board was 500x600mm in size and I then later cut out four strips with a jig-saw and utility knife for the sides of the box. Later, the sides were glued together with a hot-glue gun to give it a platform and a place to high the Arduino circuit from the roads and rails. A screw/nail would not have been as effective as the MDF is too thin for even the smallest screws. Some hard foam was stuck to the MDF with PVA glue in the corners to help with structural rigidity.

I created a map for the railway crossing simulation on an A3 sheet of paper which will be stuck on the top of the box. Holes were drilled through the paper and through the board to allow cables to come through for the boom-gates and traffic lights.

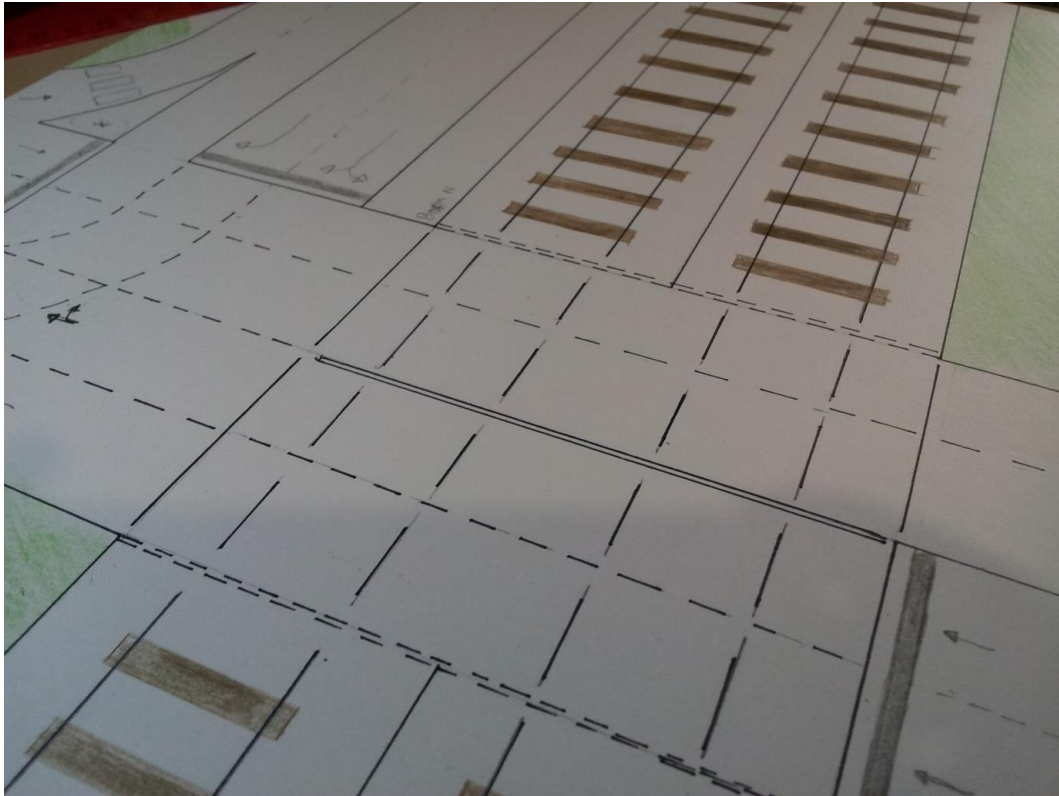


The rest of the components (eg: Arduino micro-controller, breadboard, battery) are stored underneath the board and hidden away. I put a piece of paper over the components held by Blu-Tack for easy removal.

Overall, this was a successful build and everything worked out accordingly.

ISSUES (MATERIALS)

Although there weren't any major issues with the overall build, the map did require re-adjusting to accompany the dual boom-gates and the array of traffic lights. I didn't want to put lights and servos too close together so the layout required some revising.



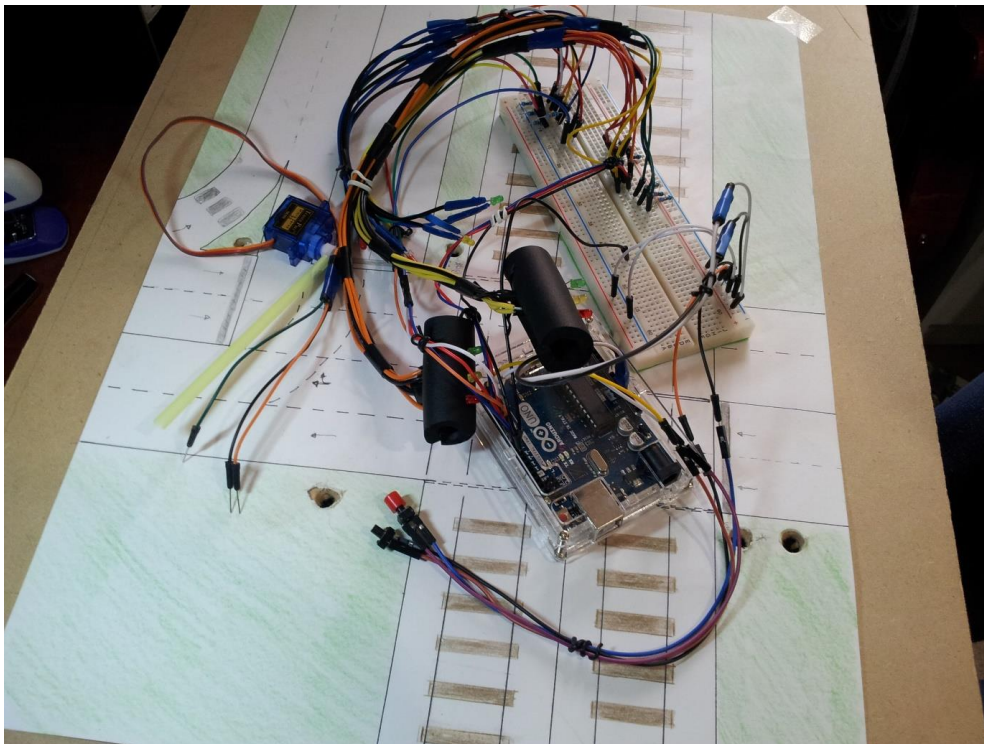
ELECTRONICS

Getting the electronics sorted out was hard work as there was lots of soldering for the LEDs. The cable-management had to be thought out according to my map as I didn't want cables tangling together.

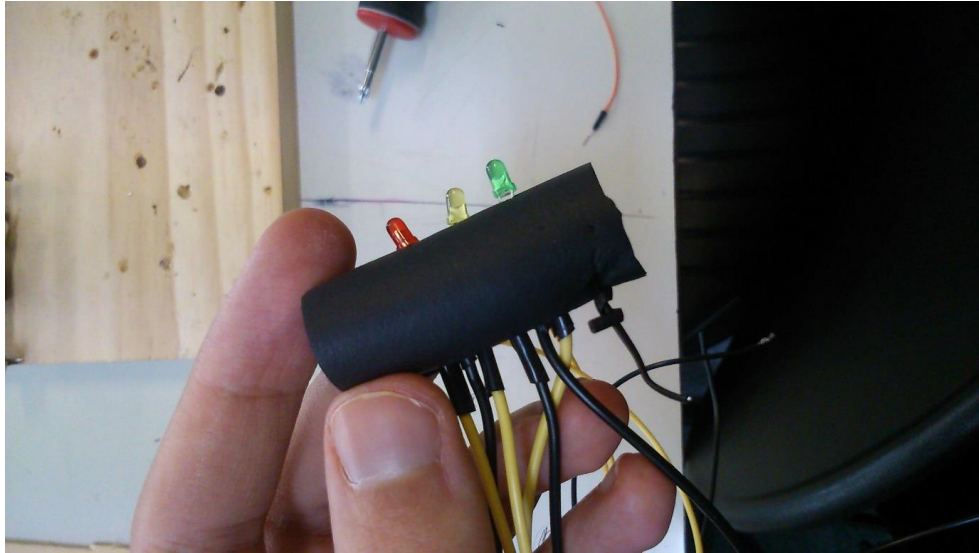
The schematic can be found further down on page 24.

The large breadboard was a quite versatile as I could layout the servos, switches and LEDs far apart to allow for less clutter. The breadboard pins weren't the best in quality so some pins often fell out and needed refitting.

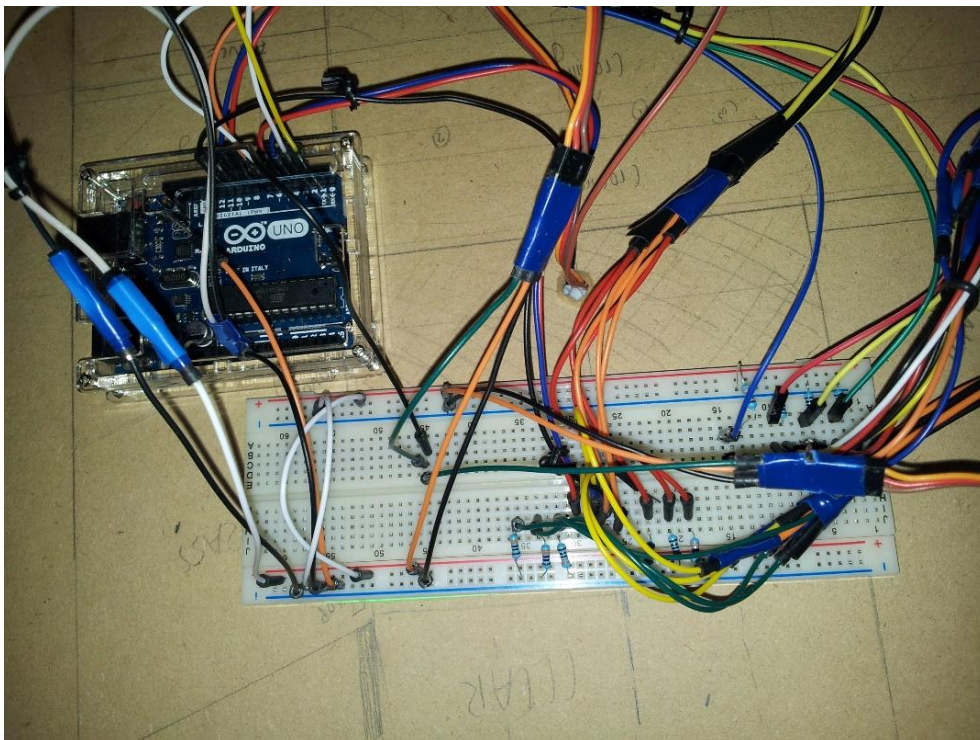
Here are the components before they were installed into the back of the board. Many of the LEDs are bunched up together to form the traffic lights. I didn't see a need to solder the components separately (creating their own ground connection and being free from the breadboard because it may be hard to diagnose if an issue arises).



A photo which I took when I had soldered a set of LEDs and put them in the foam as a traffic light. I wanted to keep the wire colours consistent so I made sure all the ground legs were using the black wire to avoid confusion later.

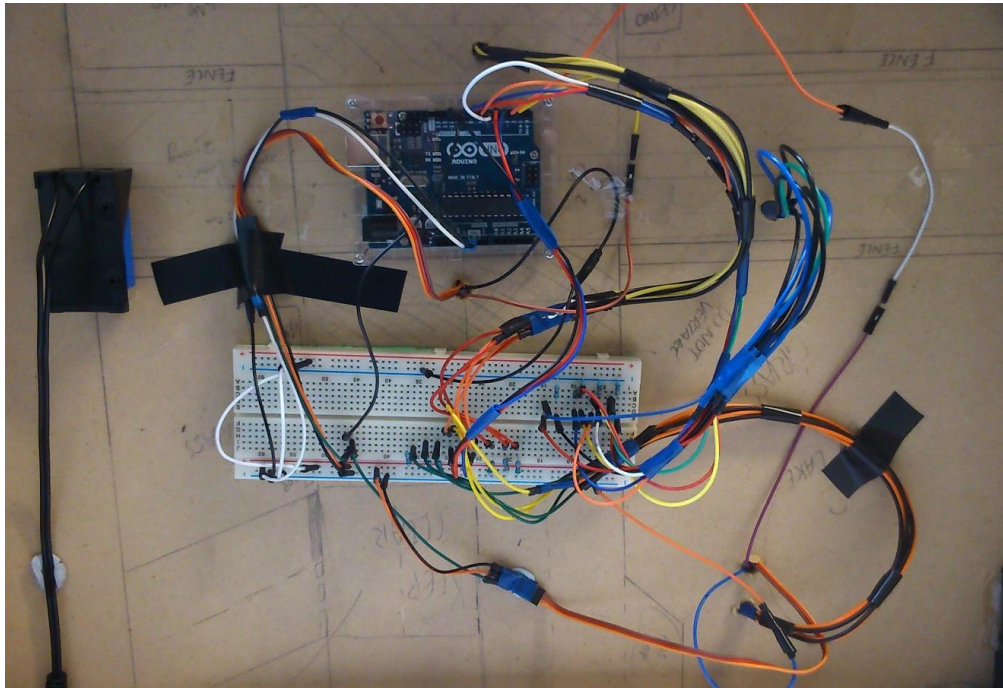


Here is a photo taken from the back of the model board as I was feeding the wires through the board. Both the Arduino and the breadboard is held with Blu-Tack.



The final layout of the railway crossing simulation.

Many wires were secured down with electrical tape and Blu-Tack . A piece of a A3 paper was then later put over it to catch any loose components from falling out. It can be easily removed as it is secured with Blu-Tack.




ISSUES (ELECTRONICS)

There aren't many issues with the electronics as the circuitry is simple. There are only three minor issues currently however they don't affect the performance or goal of the simulation in any way.

- Some jumper pins fall out due to not being secured properly on the breadboard.
- "Ground to ground" or "5V to 5V" jumper cables sometimes fall out which need to be refitted at an angle.
- The cable-management could be better however it works for the most part.

[illegible]

Program: www.github.com/berkiyo/railway-crossing-simulation (All rights reserved)

All the  go to Arduino GND

PROGRAMMING

Programming is a big part of the simulation therefore it is essential that it is thought out correctly with a pseudo-code. Due to the limited scope of the project, the pseudo-code is made accordingly.

PSEUDOCODE

Legend:

- NORTH/SOUTH = referring to the road or the traffic lights
- EAST = referring to the road or the traffic lights
- TRAIN-TRIGGER = When a train is approaching then this routine starts.
- EAST-TRIGGER = When there is traffic on the EAST road then this routine starts.

Condition 1:

- No trigger is pressed
- NORTH/SOUTH road stays green and EAST road stays red
- *This simulates that there is no traffic on the east road and no incoming train.*

Condition 2:

- TRAIN-TRIGGER is pressed
- NORTH/SOUTH lights go red depending on routine
- EAST left turn goes red however cars that want to turn right can as it will be green for them.
- *This simulates what happens once a train is approaching the intersection.*

Condition 3:

- EAST-TRIGGER is pressed
- NORTH/SOUTH lights go red
- All green for EAST road
- *This simulates what happens when there is traffic on the east road.*

Condition 4:

- EAST-TRIGGER is pressed

- EAST-TRIGGER is active
- TRAIN-TRIGGER is pressed
- TRAIN-TRIGGER overrides EAST-TRIGGER
- EAST-TRIGGER goes through its routine then followed by TRAIN-TRIGGER
- *This simulates what happens when the east-trigger routines undergo it's operation however the train-trigger was pressed during that routine. This works because I am making use of the interrupt signals with the switches.*

Perhaps the programming was the main star of the overall project since it required most of the thinking that was needed to simulate the various signals and traffic conditions. It was not a straight forward as it seems here. This was the part where trail and error and research was most extensive.

After completing the flow of various signals and conditions I was able to create the program and then later upload that to the micro-controller within Arduino.

I've used notation to reference the pin numbers with the equivalent variable names within the program. For example: the pins on the micro-controller board should match with the variable names used in the pseudo-code.

There were times where I had to download extra libraries as part of the programming to make things function. For example a library was required for the "servos" to make them work (note → #include <Servo.h>)

I have also uploaded the program along with the schematics (in PDF for printing purposes)

I believe the programming language represents the C programming language closely.

A copy of the program used is uploaded to a free online site for easy download. The location and name used for this is Arduino sketch (.ino) at:

www.github.com/berkiyo/railway-crossing-simulation

```
#include <Servo.h>
Servo boomgatesServo; // Servo

int pin[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13};
int RED = 0; int GREEN =1;  int OFF= 2;
int trainTrigger = pin[3];
int eastTrigger = pin[2];

int northsouthGreen = pin[11];
int northsouthYellow = pin[4];
int northsouthRed = pin[5];
int eastGreen = pin[6];
int eastYellow = pin[7];
int eastRed = pin[8];
int eastLeftRed = pin[9];
int boomgates = pin[10];

int northsouthTimer = 0;
int eastTimer = 0;
int northsouthStatus = RED; //red =0, green = 1
int eastStatus = RED; //red =0, green = 1

int eastTriggerState = 0;
int trainTriggerState = 0;

void setup() {

  pinMode(eastTrigger, INPUT);
  pinMode(trainTrigger, INPUT);
  pinMode(northsouthGreen, OUTPUT);
  pinMode(northsouthYellow, OUTPUT);
  pinMode(northsouthRed, OUTPUT);
  pinMode(eastGreen, OUTPUT);
  pinMode(eastYellow, OUTPUT);
  pinMode(eastRed, OUTPUT);
  pinMode(eastLeftRed, OUTPUT);
  pinMode(boomgates, OUTPUT);

  setnorthsouthGreen(northsouthGreen, northsouthYellow, northsouthRed);
  seteastRed(eastGreen, eastYellow, eastRed);

  digitalWrite(eastTrigger, HIGH);
  digitalWrite(trainTrigger, HIGH);
  boomgatesServo.write(0); // 0 degrees
  Serial.begin(9600); //for debugging
  boomgatesServo.attach(boomgates);
  attachInterrupt (digitalPinToInterrupt (trainTrigger),
traininterruptSensor, CHANGE); // attach interrupt handler
```

```

    attachInterrupt (digitalPinToInterrupt (eastTrigger),
eastinterruptSensor, CHANGE); // attach interrupt handler
}

void loop() {

    northsouthTimer++;

    if (eastTimer > 0) {
        eastTimer--; //decrements
    }

    Serial.print("East Trigger State: ");
    Serial.println(eastTriggerState);

    checkeastTrafficSensors();
    checkTrainSensors();
    ourdelay(1000);
}

int checkeastTrafficSensors() {

    if (eastTriggerState == 1) {
        if (northsouthTimer > 5) { //5 seconds for north lights

            northsouthgreentored(northsouthGreen,
northsouthYellow, northsouthRed);
            ourdelay(1000);
            eastredtogreen(eastGreen, eastYellow, eastRed);
            Serial.println("East Traffic Active! (5 seconds)");

            ourdelay(5000);
            digitalWrite(eastTrigger, HIGH);
            eastgreentored(eastGreen, eastYellow, eastRed);
            delay(1000);
            if (trainTriggerState != 1) {
                ourdelay(1000);
                northsouthredtogreen(northsouthGreen,
northsouthYellow, northsouthRed);
            }

            northsouthTimer=0;
            eastTriggerState=0;

        }

    }

}

int ourdelay(int timer) {
    int i;
    for (i=0; i<timer; i++) {
        if (trainTriggerState == 1) break;
        delay(1);
    }
}

int checkTrainSensors() {

```

```

int pos = 0;
if (trainTriggerState == 1) {

    if (northsouthStatus == GREEN)
        northsouthgreentored(northsouthGreen,
northsouthYellow, northsouthRed);
    if (eastStatus == RED) {
        delay (1000);
        eastredtogreen(eastGreen, eastYellow, eastRed);
    }
    seteastleftarrow(RED);

    for(pos = 0; pos <= 90; pos += 1) // goes from 0 degrees to
180 degrees
    {
        boomgatesServo.write(pos); // in steps of 1 degree // tell servo
to go to position in variable 'pos'
        delay(15); // waits 15ms for the
servo to reach the position
    }
    northsouthTimer = 0;
    delay (5000);
    trainTriggerState = 0;
    digitalWrite(boomgates,HIGH);
    boomgatesServo.write(0);
    delay (1000);
    seteastleftarrow(OFF);
    eastgreentored(eastGreen, eastYellow, eastRed);
    delay(1000);
    northsouthredtogreen(northsouthGreen, northsouthYellow,
northsouthRed);

}

}

// Interrupt Service Routine (ISR)
void eastinterruptSensor ()
{
    int TriggerState = digitalRead(eastTrigger);
    if (TriggerState == LOW) {
        Serial.print("INTAeasttriggerstate: ");
        Serial.println(eastTriggerState);
        eastTriggerState = 1;
        delay(1000);
    }

} // end of eastSensor

void traininterruptSensor ()
{
    int TriggerState = digitalRead(trainTrigger);
    if (TriggerState == LOW) {
        Serial.print("INT Train Trigger: ");
        Serial.println(trainTriggerState);
        trainTriggerState = 1;
        delay(1000);
    }
}

```

```

} // end of eastSensor

// Functions
int seteastleftarrow(int LEFTARROW) {
    if (LEFTARROW == RED)
        digitalWrite(eastLeftRed, HIGH);
    if (LEFTARROW == OFF)
        digitalWrite(eastLeftRed, LOW);
}

int setnorthsouthGreen(int green, int yellow, int red) {
    digitalWrite(green, HIGH);
    digitalWrite(yellow, LOW);
    digitalWrite(red, LOW);
    northsouthStatus = GREEN;
}

int seteastRed(int green, int yellow, int red) {
    digitalWrite(green, LOW);
    digitalWrite(yellow, LOW);
    digitalWrite(red, HIGH);
    eastStatus = RED;
}

int northsouthgreentored(int green, int yellow, int red) {
    digitalWrite(green, LOW);
    delay(200);
    digitalWrite(yellow, HIGH);
    delay(1000);
    digitalWrite(yellow, LOW);
    delay(200);
    digitalWrite(red, HIGH);
    northsouthStatus = RED;
}

int northsouthredtogreen(int green, int yellow, int red) {
    digitalWrite(red, LOW);
    delay(200);
    digitalWrite(yellow, HIGH);
    delay(1000);
    digitalWrite(yellow, LOW);
    delay(200);
    digitalWrite(green, HIGH);
    northsouthStatus = GREEN;
}

int eastgreentored(int green, int yellow, int red) {
    digitalWrite(green, LOW);
    delay(200);
    digitalWrite(yellow, HIGH);
    delay(1000);
    digitalWrite(yellow, LOW);
    delay(200);
    digitalWrite(red, HIGH);
    eastStatus = RED;
}

```



```
    }  
  
int eastredtogreen(int green, int yellow, int red) {  
    digitalWrite(red, LOW);  
    delay(200);  
    digitalWrite(yellow, HIGH);  
    delay(1000);  
    digitalWrite(yellow, LOW);  
    delay(200);  
    digitalWrite(green, HIGH);  
    eastStatus = GREEN;  
}
```

ISSUES (PROGRAMMING)

Learning the Arduino programming language was fun as I developed new skills and understanding the usage of *if/else statements, functions, interrupts and creating timers*.

Some of the challenges I faced were using the Arduino interrupt pins as there were only two of them for the UNO R3. This meant that we could only implement two switches for simulating two different scenarios. This was just enough and satisfied our design project requirements.

Here are some of the issues that happened in the programming section:

- Finding a suitable pin for the use as an interrupt signal.
- Semi-colons “;” were required after every statement otherwise the program compilation did not finish and gave an error. So, never forget the semi-colon after each statement!

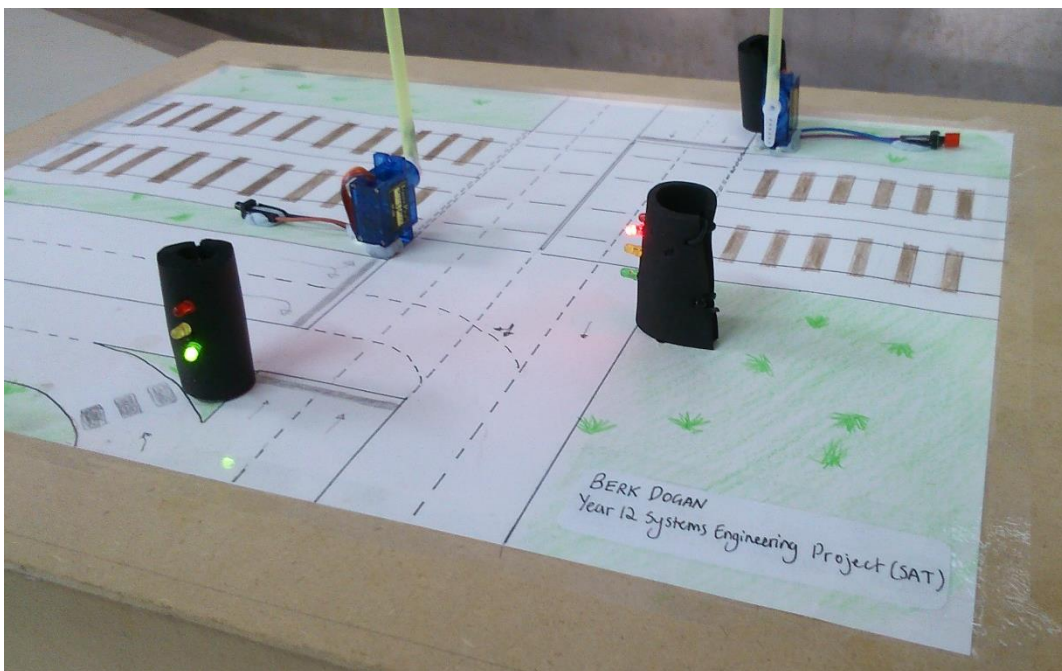
Although I don't have an error log as they were all fixed in the programming stage. It would have made impracticable to record every signal error log.

TESTING

TEST #1

This is a test of plugging in the power source and making sure the Arduino loads the default routine.

- Plug in 9V battery (**PASS**)
- Green light on NORTH/SOUTH road and red on EAST road (**PASS**)

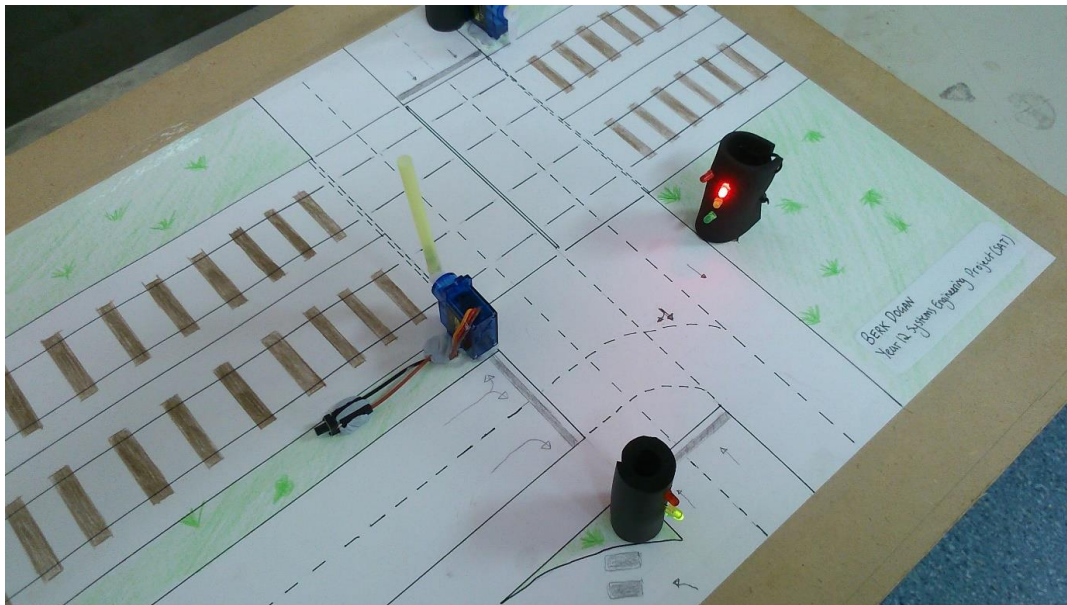


TEST #2

This is a test of pressing the EAST-TRIGGER, making sure it goes through the routine and reverts to default automatically.

- Plug in 9V battery (**PASS**)
- Green light on NORTH/SOUTH road and red on EAST road (**PASS**)
- Press EAST-TRIGGER (**PASS**)
- EAST-TRIGGER routine goes through (**PASS**)

- Back to Green light on NORTH/SOUTH road and red on EAST road automatically **(PASS)**

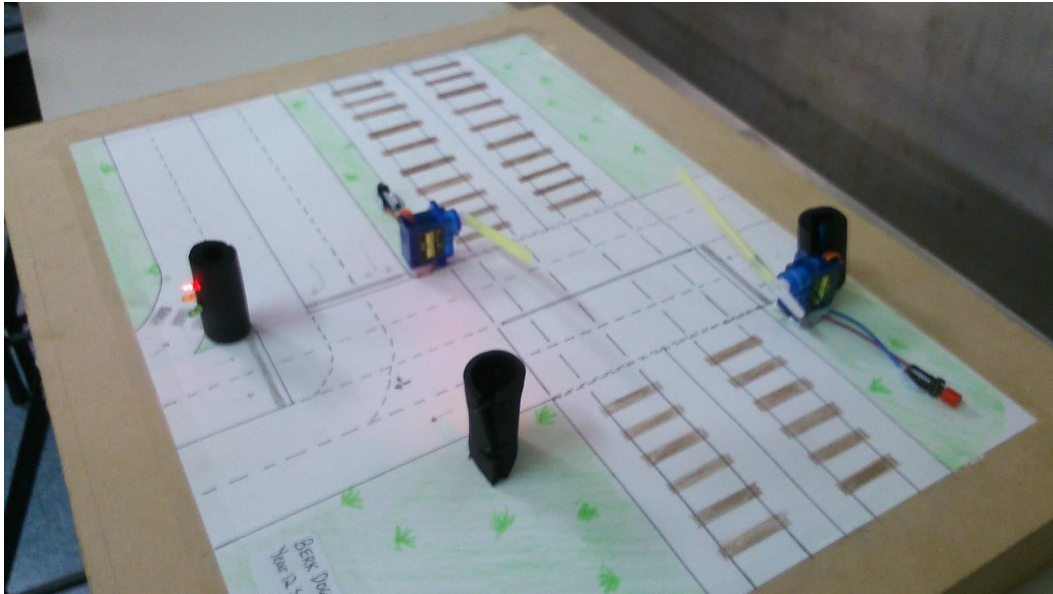


TEST #3

This is a test of pressing the TRAIN-TRIGGER, making sure it goes through the routine and reverts to the default automatically.

- Plug in 9V battery **(PASS)**
- Green light on NORTH/SOUTH road and red on EAST road **(PASS)**
- Press TRAIN-TRIGGER **(PASS)**
- TRAIN-TRIGGER routine goes through (boom-gates come down) **(PASS)**

- Back to Green light on NORTH/SOUTH road and red on EAST road automatically (**PASS**)



TEST #4

This is a test of pressing the EAST-TRIGGER and then momentarily pressing TRAIN-TRIGGER. It should go through the TRAIN-TRIGGER (as an interrupt) routine and then revert to the default automatically.

- Plug in 9V battery **(PASS)**
- Green light on NORTH/SOUTH road and red on EAST road **(PASS)**
- Press EAST-TRIGGER **(PASS)**
- EAST-TRIGGER begins (the sequence begins) **(PASS)**
- TRAIN-TRIGGER is momentarily pressed (approximately one second after EAST-TRIGGER is pressed) **(PASS)**
- TRAIN-TRIGGER overrides EAST-TRIGGER routine and all lights go red (except for EAST right turn) **(PASS)**
- TRAIN-TRIGGER routine goes through (boom-gates come down) **(PASS)**
- Back to Green light on NORTH/SOUTH road and red on EAST road automatically **(PASS)**

EVALUATION

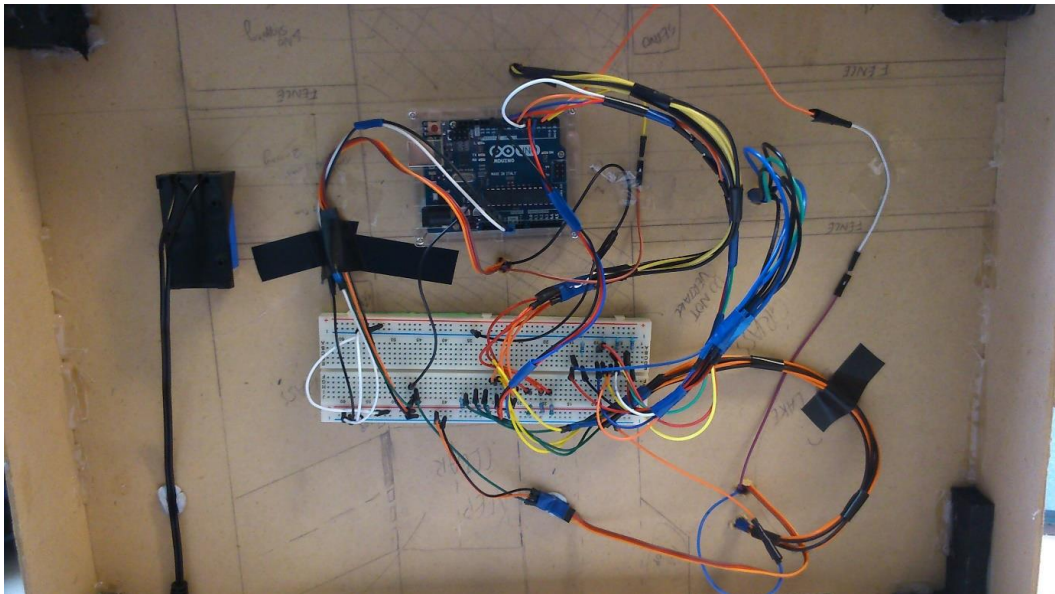
PLANNING

The timeline is a fundamental part of the project as it enables the user to meet deadlines and plan ahead. The timeline was very effective when followed closely and it gives plenty of time to complete all tasks and any issues that may arise in the process.

IMPROVEMENTS

Although the project is now finished, there are a few things I would like to improve to the aesthetics of the project.

- Use a small board, closer to the size of an A3 sheet of paper. As it can be seen in the image below, there was quite a lot of empty space under the box so a smaller box would have been good too.



- Paint the box in a singular colour so it looks nicer.
- Solder the components directly to avoid using a breadboard.

EVALUATION OF COMPONENTS

Arduino Uno R3 (assembled with components)

- The micro-controller behaves as it should and there are no issues.

SG90 Servos (assembled and plugged into breadboard)

- The servos behave normally and work as intended. There is sometimes a ticking sound however that can be fixed in the Servo.h file. This will stop the chattering with the servo and the micro-controller. There are no issues.

Overall structure of the box

- The materials are holding the box well and there are no signs of degradation or damage. There are no issues.

Buttons

- Buttons work as intended. More information on how they work on pages 34 to 37. There are no issues.