[redacted]

Kurs: [redacted]

Fachlehrer/in: [redacted]

Schuljahr: [redacted]

Abgabetermin: _____

Thema der Facharbeit:

Verbesserung einer Ampel im Stadtgebiet Detmold

Die Abgabe der Facharbeit ist fristgerecht erfolgt.

_____ _____

(Unterschrift Kurslehrer/in) (Unterschrift Jahrgangsstufenleitung)

## Inhaltsverzeichnis

Inhaltsverzeichnis ..... 2
1 Einleitung ..... 3
2.0 Problemfindung, -auswahl und Lösungsansätze ..... 3
2.1 Begriffe Klärung ..... 3
2.2 Probleme mit LSAs ..... 3
2.2.1 Ausfall ..... 3
2.2.2 Grüne Welle ..... 3
2.2.3 Fußgänger Probleme ..... 4
2.2.4 Fahrradfahrer Probleme ..... 4
2.2.5 Autofahrer Probleme ..... 4
2.3 Probleme und wie sie gelöst werden können ..... 4
2.3.1 Problemlösung Ausfälle ..... 5
2.3.2 Problemlösung Grüne Welle ..... 5
2.3.3 Problemlösung Fußgänger Grünzeiten ..... 5
2.3.4 Problemlösung Rot-Grün Zeiten ..... 5
2.3.5 Weitere Probleme ..... 5
3 Umsetzung ..... 5
3.1 Generelle Voraussetzungen ..... 5
3.1.1 Auswahl der Kreuzungen. ..... 6
3.1.2 Auswahl der Probleme ..... 6
3.2 Das Grobe Modell ..... 7
3.3 Technische Umsetzung ..... 7
3.3.5 Exkurs: Arten von Ampelsteuerungen ..... 7
3.4 Bau ..... 8
3.5 Die Programmierung ..... 9
3.5.1 Exkurs: Grober Aufbau C++ ..... 9
3.5.2 Setups Erläuterung ..... 9
3.5.3 Loop Erleuterung ..... 10
3.5.4 Umsetzung der Probleme ..... 10
4 Fazit ..... 10
Literaturverzeichnis ..... 12
Anhang: ..... 12
Code ..... 12
Abbildungsverzeichnis ..... 35

## 1 Einleitung

- Rot, Gelb, Grün, das sind die Farben der Ampel. Das weiß jedes Kind, was aber nicht jedes Kind weiß, ist, wie sie funktionieren und was es für Probleme gibt. Diese Facharbeit handelt davon und über die informatische Verbesserung von einer Ampel im Stadtgebiet Detmold handeln. Ich werden Probleme und Lösungen sammeln, eine Ampel aus dem Stadtgebiet auswählen, dies dann mit einem Arduino nachbauen und verbessern.

## 2.0 Problemfindung, -auswahl und Lösungsansätze

In diesem Abschnitt werden die Probleme der Ampeln behandelt und im Anschluss ausgewählte Probleme gelöst.

### 2.1 Erklärung der Klärung

Mit dem Wort Ampel ist umgangssprachlich eine Lichtsignalanlage (kurz LSA), Lichtzeichenanlage (LZA) oder Wechsellichtzeichenanlage gemeint, die zur Steuerung von Straßenverkehr dient.

### 2.2 Probleme mit LSAs

LSAs haben Probleme manch Problem sind schlimmer als andere. Hier werden ich einige davon finden und analysieren. Die folgenden Daten dazu kommen aus Stichproben von der Webseite Traffic Check der Stadt Graz *(https://urlzs.com/4suZM 12.2.2022)*. Eine Anfrage zu einer Auswertung oder den Daten in einer besseren Form blieb unbeantwortet.

#### 2.2.1 Ausfall der Anlagen

LZAs können aus vielen Gründen ausfallen. Zum Beispiel kann der Strom ausgefallen sein oder bauliche Schäden nach einem Autounfall können dieses auslösen. Auch bei Baumaßnahmen wird die Stromversorgung einer Ampelanlage nicht selten beschädigt oder ausgeschaltet.

#### 2.2.2 Grüne Welle

Die „Grüne Welle“ ist die spezielle Schaltung der LSAs eines Straßenzuges, um möglichst alle Grünphasen beim Durchfahren zu ermöglichen *(Abbildung 1).* Eine solche Schaltung hat viele Vorteile, wie das Schonen der Nerven von Autofahrern, Verringerung der Emissionen durch die verkürzten Wartezeiten und die Vermeidung von Stop-and-Go.

*(Brilon; Wietholt; Wu Dezember 2007 , S.62)* Die Grüne Welle kann häufig nicht problemlos umgesetzt werden, weil es meist unmöglich ist, absolut alle LSAs perfekt zu

schalten. Dieses liegt an der Zeit, die gebraucht wird, um eine Strecke zurückgelegen. So kann es zwischen zwei LSAs problemlos funktionieren und zwischen zwei anderen Kreuzungen nicht, weil man im Stadtverkehr möglichst eine konstante Geschwindigkeit halten möchte.
Theoretisch ist es aber möglich, „Grüne Wellen“ für alle Richtungen und den jeweiligen Querverkehr einzurichten.

### 2.2.3 Fußgänger Probleme

Fußgänger beschweren sich oft über die Wartezeiten und die Länge der Grünphasen. Auch liegen Beschwerden vor, dass die Barrierefreiheit nicht immer gegeben ist. Ein großer Punkt bei den Grünzeiten ist, dass nicht jeder Fußgänger annähernd gleich schnell ist und somit deutlich unterschiedliche Zeiten brauchen.

### 2.2.4 Fahrradfahrer Probleme

Radfahrer queren Kreuzungen auf zwei Wege. Entweder sie nutzen die Fußgängerüberwege oder die Verkehrsflächen. Wenn sie die Fußgängerüberwege befahren, haben sie die gleichen Probleme wie die Fußgänger. Fahren sie nun jedoch auf der Straße, so bemängeln sie, dass es für sie schlechte Sicht und Sicherheitsbedingungen gibt. Aber noch öfter wird sich über fehlende Platzverhältnisse oder Konfliktpotenzial beschwert.
Dieses hat ihren Ursprung darin, dass die LSAs früher, ohne die Fahrradfahrer zu berücksichtigen, geplant und gebaut wurden und dann nachträglich Fahrradstreifen und weiteres, wie spezielle Lichtzeichen, hinzugefügt wurden, ohne große bauliche Änderungen durchzuführen. Dieses hat zur Folge, dass Fahrradstreifen oft nicht existent sind und wenn doch, so sind diese meist nur enge Markierungen auf dem Asphalt.

### 2.2.5 Autofahrer Probleme

Diese Gruppe von Verkehrsteilnehmern beschwert sich wie alle anderen, über die Rot- und Grün-Phasen. Spezifisch aber bei kleinen LSAs über die Platzverhältnisse und bei großen Anlagen über die Übersichtlichkeit. Generell wird aber auch beanstandet, dass manche LSAs ein erhöhtes Konfliktpotenzial haben oder die Sicht eingeschränkt ist.

## 2.3 Probleme und wie sie gelöst werden können

Es werden nun einige Probleme ausgewählt und geschaut, ob diese zu lösen sind und wie der Lösungsweg aussehen könnte.

### 2.3.1 Problemlösung Ausfälle

Probleme mit Ausfällen kann ich nicht lösen, hierzu können jedoch präventive Maßnahmen getroffen werden, wie baulicher Schutz oder Notstrom, welche jedoch außerhalb des Themenbereiches liegen.

### 2.3.2 Problemlösung Grüne Welle

Die Grüne Welle kann ich realisieren, indem ich eine optimale Stadt plane. In der Facharbeit möchte ich mich jedoch eher auf eine einzelne Kreuzung konzentrieren.

### 2.3.3 Problemlösung Fußgänger Grünzeiten

Dieses Problem kann man damit lösen, dass die Grünzeiten dynamisch angepasst werden können. Das kann beispielsweise über Detektoren in den Überwegen gelöst werden.

### 2.3.4 Problemlösung Rot-Grün Zeiten

Rot-Grün Zeiten kann man mit einer dynamischen Schaltung optimieren. Wie genau wird später besprochen.

### 2.3.5 Weitere Probleme

Probleme wie Sicht und Sicherheitsbedingungen, fehlenden Platz oder Konfliktpotenzial können durch bauliche Maßnahmen gelöst werden.

## 3 Umsetzung

Nun geht es an die Umsetzung der Facharbeit, die Auswahl der Probleme und der Kreuzung. Schlussendlich werde ich dann die ausgewählten Probleme der ausgewählten Kreuzung gelöst.

### 3.1 Generelle Voraussetzungen

Bei diesem Projekt soll es sich um die Umsetzung und Verbesserung einer LSA handeln. Dazu wurde ein Modell erbaut, um die Verbesserungen anschaulich zu beschreiben. Die elektronische Funktionalität soll über einen sogenannten Arduino passieren. Hierbei handelt es sich um einen einsteigerfreundlichen Microcontroller, welcher über die Programmiersprachen C++ programmiert wird. Am wichtigsten ist jedoch, dass das Modell auch StVo konform ist.

### 3.1.1 Auswahl der Kreuzungen.

Um nun genaueres zu planen, hatte ich im Vorfeld einen Antrag nach dem Informationsfreiheitsgesetz NRW an Jochen Detering (zuständiger für die LSAs im Stadtgebiet Detmold) gestellt, in welchem ich Daten über 5 verschiedene Lichtsignalanlagen im Stadtgebiet Detmold angefragt habe. Die Auswahl dieser 5 Anlagen ist durch meinen täglichen Schulweg, Problemampeln und generellem Interesse entstanden. Um nun abzuwägen welche dieser LSAs ich daraufhin nachbauen und verbessern möchte, haben ich die Vor- und Nachteile in einer Tabelle zusammengefasst:

| Kreuzung | Karte | Informationen | Baulich | Probleme |
|---|---|---|---|---|
| Paderborner Str. - Hornsche Str. | 2 | 4 | 2 | 3 |
| Röntgenstraße - Paulinenstraße - Lagesche Str. - Lemgoer Str. | 0 | 2 | 2 | 2 |
| Bielefelder Str. - Hiddeser Str. - Heidenoldendorfer Str. | 3 | 3 | 3 | 2 |
| Theodor-Heuss-Straße - Friedrich-Ebert-Straße - Hindenburgstraße | 1 | 1 | 1 | 2 |
| Denkmalstraße - Paderborner Str. – Externsteinestraße | 1 | 1 | 1 | 1 |

Schlussendlich ist die Entscheidung auf die Kreuzung Paderborner Str. - Hornsche Str. gefallen, weil ich zu dieser viele Informationen sammeln konnte. Beispielsweise hat das PDF der Karte verschiedenen Ebenen, welche sich einzeln ein- und ausblenden lassen, wodurch ich das Modell deutlich übersichtlicher gestalten kann. Auch habe ich hierbei Variablenlisten und Programmdiagramme. Doch das Wichtigste ist, dass ich bei dieser Ampel die Auslastungen habe, welches mir ermöglicht, die verschiedenen Verkehrsbelastungen einzubeziehen.

### 3.1.2 Auswahl der Probleme

Ich fragte diese Lichtsignalanlagen ursprünglich ab, weil an dieser morgens an den Wochentagen fast immer kilometerlange Staus entstehen. Also ist dieses ein Problem, welchem ich mich annehmen werde. Des Weiteren soll dann auch die Grünzeit der Fußgänger und Radfahrer optimiert werden.

## 3.2 Modell bauen

Bei der Hülle gibt es verschiedene Voraussetzungen, beispielsweise soll das Modell gut transportierbar sein, die Technik verstecken und schützen, aber auch soll auf dem Deckel sich das Modell befinden

Bei dieser Hülle ist dann schnell die Entscheidung auf einen Karton gefallen, da dieser genug Oberfläche für das Modell bietet. Dieser wurde dann etwas zurückgeschnitten, um ihn zu verkleinern. Um den Schutz zu gewährleisten, und eine stabile Grundlage für den Bau zu haben wurden der Karton teils mit Holz ausgekleidet, wie man auf der *Abbildung 3* sieht

## 3.3 Technische Umsetzung

Bei der technischen Umsetzung der Ampel gibt es verschiedene Limitierungen. Beispielsweise die Anzahl an steuerbaren Ausgängen des Microcontroller, welche sich bei dem Arduino MEGA 2560 auf 58 beläuft. Ich habe den Microcontroller ausgewählt, da dieser die Anforderungen - wie genügend Ein- und Ausgänge - erfüllt

Das Problem der Grünzeiten der Fußgänger möchte ich lösen, indem ich Trittplatten in die Fahrbahn setze, um herauszufinden, wann und wie viele Fußgänger die Fahrbahn betreten und auch wieder verlassen haben. Bei großen Gruppen bleibt so auch länger grün.

Das Problem mit der Optimierung für große Verkehrslasten ist ein deutlich größeres und komplexeres Problem. Weil die Schaltzeiten für jeden Verkehrsteilnehmer schon sehr optimal sind.

Ich werde auf die Nutzung einiger Taster verzichten, weil ich diese nicht benötige und dieses alles nur noch komplexer machen würde.

### 3.3.1 Exkurs: Arten von Ampelsteuerungen

Es gibt viele verschiedene Arten LSAs zu steuern. Die wohl einfachste ist eine komplett „dumme“ Ampel, eine Festzeitsteuerung. Hierbei läuft die Ampel ihr Programm, wechselt von Rot nach Gelb nach Grün nach Gelb und wieder von vorne. Dieses passiert immer mit den gleichen Zeiten und dabei empfängt sie keine Daten. Sie ist nicht verkehrszeitenabhängig und verursacht somit häufig Staus. Zum Glück findet man solche Steuerungsmethoden meist nur noch in alten mobilen Ampeln, z.B. Baustellenampeln. Optimaler hingegen ist eine verkehrsabhängige Ampel. Dieses kann ihre Grünzeiten verlängern, um mehr Verkehr schneller abfließen zu lassen. Bei diesen gibt es nun zwei fundamentale Typen. Die eine bestimmt frei ihren Ablauf. Die andere hat vorgefertigte Programme, zwischen welchen die Ampel dann frei hin und her schalten kann. Zwischen der festzeitgesteuerten und der verkehrsabhängigen,

programmabhängigen Steuerungen gibt es auch Hybride: Die Zeitsteuerung. Bei diesem werden zu spezifischen Zeiten die Programme gewechselt. Bei den Steuerungstechniken gibt es beliebig viele Abstufungen. Es gibt auch modernere Steuerungsmethoden wie das kooperative Steuerverfahren, bei welchem die Position und Geschwindigkeit der heranfahrenden Autos in die Berechnung eingehen. Oder Verlustzeit, dieses Steuerverfahren sammelt Verlustzeit sich, wenn ein Fahrzeug beim Durchfahren abgebremst wird. Dieses wird dann am Ende mit den Grünphasen verrechnet *(https://tud.qucosa.de/api/qucosa%3A26098/attachment/ATT-0/?L=1 11.4.2022)* Die Stadt Detmold benutzt auf allen 5 angefragten Ampeln eine verkehrsabhängige Plansteuerung.

In meiner Facharbeit werde ich versuchen, eine Kombination aus Verlustzeit, kooperativen und freier Ablauf-Steuerung zu programmieren.

## 3.4 Bau

Für den Bau hatten ich schon zuvor einen Kasten gebaut und verstärkt. Auf diesen wurde dann der Plan der LSA aufgeklebt (*Abbildung 10*). Um nun die verschiedenen Verkehrswege zu „Steuern“, habe ich für die Fußgänger und Autofahrer aufgezeichnet (*Abbildung 9*), wo die verschiedenen Signale stehen sollen und dieses dann mit einem 5mm Bohrer gebohrt. Aufgrund eines Plastik Ringes an der Seite, ließ sich die aber nicht bis zur gewollten Tiefe einstecken. Diesen habe ich daraufhin bei jeder LED abgekniffen und dann mit Heißkleber eingeklebt.
Die Katode(-Pol) der LED habe ich dann mit allen anderen der Gruppe verlötet und an die Anode(+Pol) habe ich einen 220 Ohm Widerstand angelötet, um das Durchbrennen der LED bei 5V zu verhindern (*Abbildung 8*).
Für die Taster habe ich zuerst die Pappe an den Stellen, an welchen ein Taster sein sollte, weggeschnitten und die Ecken mit einem 3mm Bohrer gebohrt (Abbildung 7). In dieses werden dann die Beine der Taster gesteckt. Danach wurden Die LEDs und Taster mit dem Arduino verkabelt. Bei der Kabelfarbwahl habe ich mich für Rot Gelb Grün für die entsprechenden LEDs der Ampeln entschieden. Weißgrau ist Minus oder Plus, Blau sind Verkehrstaster und Lila sind die Fußgängertaster. Die Kabel habe ich dann mit Kabelbindern und 3D gedruckte Kabelhalterungen die Kabel ordentlich verlegt (Abbildung 3). Der Microcontroller wurde in einem von mir in 3D gedruckten Halter fixiert. Bei der Pin-Belegung habe ich die Outputs, sprich die LEDs, auf die PWM und analog Pins (Was die Pins besonderes können ist absolut irrelevant)gesteckt (Abbildung 3). Die Inputs werden dann auf die Digitalpins gesteckt. Dieses hat keinen besonderen Grund bis auf, dass die Inputs und Outputs ordentlich getrennt sind. Am Ende wurden noch 3D

gedruckte Abdeckungen für die Buttons eingeklebt, um das Modell optisch noch mehr aufzuwerten.

## 3.5 Die Programmierung

Bei der Programmierung gibt mehr vorab Entscheidungen. Als allererstes mussten grobe Design-Entscheidungen getroffen werden. Ich habe mich in für zwei Quelltext Dateien entschieden.

### 3.5.1 Exkurs Vereinfachte C++ Strukturen

Ein Programm in C++ besteht, wenn man richtig programmiert aus einer .cpp Datei und einer .hpp Datei. Die cpp Dateien beinhalten den Quelltext, also die Logik, welche für uns am relevantesten ist. Jedoch finde ich, dass die .hpp Dateien, der header, auch erwähnenswert sind. Eine .hpp. die Datei beinhalten die Deklarationen von Klassen und Variablen. Wenn eine Quelltext Datei auf die Funktionen einer anderen .cpp Datei zugreifen möchte, braucht man nur die zu der gesuchten .cpp passende.hpp mit „#include“ einfügen.
Die eine Datei wird die ganze Logik (ampel.cpp) beinhalten und die andere wird die LEDs steuern (interface.cpp).

Des Weiteren habe ich mich bei der Benennung der Variablen, an den des Planes orientiert. So heißen die Auto Ampeln K und die Fußgänger und Radfahrer – Ampeln FR. Ich habe mich nicht an den Variabellen Listen in den Daten von Herr Detering orientiert, da diese Ampeln ja wie zuvor festgestellt nach einem andern Steuerverfahren laufen.
Der Code in C++ für den Arduino hat eine „void setup()“ und eine „void loop()“ Die setup wird zu Beginn des Scripts einmal ausgeführt. Die loop hingegen läuft die ganze Zeit durch. Ist das Script unten fertig beginnt es oben wieder.

### 3.5.2 Setup Erläuterung

In dem Setup passiert kaum etwas Spannendes. Zuerst wird der serielle Monitor aufgebaut (eine Verbindung zwischen PC und Arduino, um Daten zu übertragen, häufig und hier auch zum Debuggen genutzt) und danach die erste Nachricht mit verschickt. Im Anschluss werden die Pins definiert, ob sie ein Input oder Output sind. Dieses passiert durch die ersten drei „for“ schleifen. Die Darauffolgenden „for“ schleifen setzen nur Inhalte in sonst leere Arrays. Ein Array ist quasi eine Speisekarte, man kann eine Nummer an das Array geben und dann bekommt man, dass was an der Speicherstelle steht.

### 3.5.3 Loop Erläuterung

Die Loop ist deutlich komplexer, hier werden direkt Funktionen ausgeführt. Die „TasterCheck()“ Checkt welche Taster gerade gedrückt sind und schreibt in einem Array mit, welcher der Wie häufig gedrückt wurde. Der „TasterParser()“ nimmt sich das Array von dem „TasterCheck()“ und sucht hierbei sich die wichtigen Sachen heraus und ermittelt damit den Verkehr, welcher aus den verschiedenen Richtungen kommt und welcher gerade durch die Ampel gefahren ist. Die darauffolgende Funktionen FRCheck() und FRParser() haben das gleiche Ziel wie die beiden für die Taster, nur halt für die Fußgänger.

In dem If Block wird geprüft, ob genug Zeit vergangen ist. Die einzelnen Verzögerungen werden addiert und geschaut, ob sie kleiner als millis() ist. millis ist eine interne Zeit in Millisekunden. delays(), also ein Watrpause, hält den Prozessor an und so registrieren dich dann keine Taster Events mehr. Diese Schleife würde nur einmal funktionieren, weil nachdem alle addierten Sachen einmal weniger als millis sind werden sie es immer sein, deswegen gibt es in der Addition die Variabel Relative0, sie wird jede runde auf millis() gesetzt, und ermöglicht dann das wiederholen. Am Ende jeder Runde werden danach die Zeiten neu berechnet und gesetzt, wodurch wir eine Kooperative Steuerung haben. Auch wird die Verlustzeit abgezogen. Die interface.cpp macht währenddessen eher die Animationen der LEDs.

### 3.5.4 Umsetzung der Probleme

Ich habe das Problem mit dem Stau versucht zu lösen, indem ich den Zeitenalgorithmus von einem planbasierten zu eine freien, kooperativen und Verlustzeitbasierten Steuerung programmiert habe. Dieses wird im Anfang der TimeingSet() berechnet.

Auch ein Grüner Pfeil bei der Ampel K1 habe ich hinzugefügt um das Abbiegen zu erleichtern.

Das Problem mit den Fußgängern konnte ich leider nicht lösen, ich hatte Probleme bei der Datenverarbeitung.

## 4 Fazit

Als Fazit kann man sagen, dass der Bau gut gelungen ist und die Implementierung auch nicht schwer war, nun stellt sich nur die Frage, wieso die Stadt eine solche Schaltung noch nicht eingebaut hat. Ich werde darüber mich mit Herr Detering in Verbindung setzen. Ich habe schlussendlich einen Einblick bekommen, wie Ampeln funktionieren, was es für Unterschiede gibt, und wie man Ampeln vielleicht sogar verbessern kann. Auch wenn ein kleines Modell nicht sehr realistisches ist, hat das Modell trotzdem sein Ziel, etwas simpler zu gestalten, erfüllt.

## 4.1 Für die Zukunft

Ich habe in dieser Facharbeit gelernt, dass eine Anständige Vorplanung von Nöten ist, so hätte ich die Kabel mit 5v eine Andere Farbe gemacht als die Ground Kabel. Ich hätte auch die FR Taster durchgeschliffen, anstatt noch 3 Pins zu belegen. Oder hätte ich die FR Taster mittig auf der Straße platziert und nur einen pro Überweg verwendet. So hätte ich dann auch wahrscheinlich das Programm funktionstätig zu bekommen. ursprünglich hatte ich noch den Plan, einen eigenen Steuerungsalgorithmus geplant, diesen musste ich dann aber wegen Zeitproblemen streichen. Das Projekt hat mir auch meine Programmiertechnischen Schranken gezeigt. Aber generell würde ich das Modell wieder gleich bauen.

## Literaturverzeichnis

*https://urlzs.com/4suZM (12.2.2022)*

*https://tud.qucosa.de/api/qucosa%3A26098/attachment/ATT-0/?L=(1 11.4.2022) Oertel Brilon, Werner/Wietholt, Thomas/Wu, Ning (Dezember 2007): Empirische Absicherung innovativer Steuerverfahren für Lichtsignalanlagen imöffentlichen Straßenraum. DLR*

Daten NRW Informationsfreiheitsgesetz (24.3.2022) Jochen Detering

## Anhang:

### Code

Ampel.cpp

```
#include "Ampel.hpp"
#include "interface.hpp"


const uint8_t Detect[17] = {53, 52, 51, 50, 49, 48, 47, 46, 45, 43, 41, 39, 37, 35, 33, 31, 29};
const uint8_t FR[12] = {44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22};



uint8_t FROUT[12] = {};
uint8_t FRPressed[12] = {};
uint8_t DetectOUT[17] = {};
uint8_t DetectPressed[17] = {};

// K1
const uint16_t MaxTimeK1 = 7000;
const uint16_t DefaultTimeK1 = 1900;
uint32_t ActualDurationK1 = 40000;

// K3
const uint16_t MaxTimeK3 = 7000;
const uint16_t DefaultTimeK3 = 1900;
uint32_t ActualDurationK3 = 42000;
```

```
// K4
const uint16_t MaxTimeK4 = 7000;
const uint16_t DefaultTimeK4 = 1000;
uint32_t ActualDurationK4 = 17000;

// k
uint32_t ActualDurationK1uK3;

// Pfeil
uint32_t ActualTimePfeil = 17000;
bool AnimatPfeil = false;


// FR
const uint16_t MinFR = 2000;
const uint16_t MaxFR = 7000;

// FR 1
bool TraffiFR1 = false;
bool EnableFR1 = false;
bool AnimatFR1 = false;
uint32_t ActualDurationFR1 = 10000;
uint16_t TraffiK1 = 0;

// FR 3
bool TraffiFR3 = false;
bool EnableFR3 = false;
bool AnimatFR3 = false;
uint32_t ActualDurationFR3 = 10000;
uint16_t TraffiK3 = 0;

// FR 4
```

```
bool TraffiFR4 = false;
bool EnableFR4 = false;
bool AnimatFR4 = false;
uint32_t ActualDurationFR4 = 25000;
uint16_t TraffiK4 = 0;

const uint16_t VerlustT = 500;
const uint16_t DefaultTimePfeil = 1000;
const uint16_t TraffikNoise = 5;
const uint16_t AutoZeit = 500;
uint16_t SafeTime = 10000;
uint32_t Relative0 = 0;
uint32_t cleanFRgreen = 1000;
int Loop = 0;

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

 Serial.println("OK Lets go!");

 // output pins ***klug*** Definieren
 for (int index = 0; index < ELEMENTCOUNT(Lampen); index++)
 {
  pinMode(Lampen[index], OUTPUT);
 }

 for (int index = 0; index < ELEMENTCOUNT(Detect); index++)
 {
  pinMode(Detect[index], OUTPUT);
 }
```

```
 for (int index = 0; index < ELEMENTCOUNT(FR); index++)
 {
  pinMode(FR[index], OUTPUT);
 }

 for (int index = 0; index < ELEMENTCOUNT(DetectPressed); index++)
 {
  DetectPressed[index] = false;
  DetectOUT[index] = 0;
 }

 for (int index = 0; index < ELEMENTCOUNT(FRPressed); index++)
 {
  FRPressed[index] = false;
  FROUT[index] = 0;
 }
 setFR1(false);
 setFR3(false);
 setFR4(false);
}

void loop()
{
 TasterCheck();
 //Serial.println(TasterCheckString());
 TasterParser();
 // Serial.println(TasterParserString());
 FRCheck();
 // Serial.println(FRCheckString());
 FRParser();
 // Serial.println(FRParserString());
```

```
// ############################################## K
  if (Relative0 + SafeTime + cleanFRgreen + ActualDurationK4 + SafeTime + cleanFRgreen + ActualDurationK3  < millis() && Loop >= 3)
  {
    SafeTime = 10000;
    setK3(false);
    setK1(false);
    if (getDoneK3() && getDoneK3())
    {
      TimingSet();
    }
  }

  if (Relative0 + SafeTime + cleanFRgreen + ActualDurationK4 + SafeTime + cleanFRgreen < millis() && Loop == 2)
  {
    setK3(true);
    setK1(true);
    if (getDoneK3())
      Loop++;
  }

  if (Relative0 + SafeTime + cleanFRgreen + ActualDurationK4 < millis() && Loop == 1)
  {
    SafeTime = 2000;
    setK4(false);
    setPfeil(false);
    if (getDoneK4())
      Loop++;
  }
  if (Relative0 + SafeTime + cleanFRgreen< millis() && Loop == 0)
```

```
 {
  setK4(true);
  if (getDoneK4())
   Loop++;
 }

 if(Relative0 + ActualTimePfeil < millis() && !AnimatPfeil){
  setPfeil(true);
  AnimatPfeil = true;
 }



//########################## FR
 if (EnableFR1){
  if (Relative0 < millis() && !getFR1() && !AnimatFR1){
   Serial.println("FR1 on");
   setFR1(true);
  AnimatFR1 = true;
  }
  if (Relative0 + ActualDurationFR1 < millis() && AnimatFR1){
   Serial.println("FR1 off");
   setFR1(false);
   EnableFR1 = false;
  }
 }
 if (EnableFR3){
  if (Relative0 < millis() && !getFR3() && !AnimatFR3){
   Serial.println("FR3 on");
   setFR3(true);
   AnimatFR3 = true;
  }
```

```
   if (Relative0 + ActualDurationFR3 < millis() && AnimatFR3){
     Serial.println("FR3 off");
     setFR3(false);
     EnableFR3 = false;
   }
  }
  if (EnableFR4){
   if (Relative0 + ActualDurationK4 + SafeTime < millis()  && !AnimatFR4) {
     Serial.println("FR4 on");
     setFR4(true);
     AnimatFR4 = true;
   }
   if (Relative0 + ActualDurationK4 + SafeTime + ActualDurationFR4 < millis() && AnimatFR4){
     Serial.println("FR4 off");
     setFR4(false);
     EnableFR4 = false;
   }
  }
}

//########################################## Time Math ##########################
void TimingSet()
{
  // ######################## adaptive K Zeiten
ActualDurationK1 = DefaultTimeK1;
ActualDurationK3 = DefaultTimeK3;
ActualDurationK4 = DefaultTimeK4;

if (TraffiK1 > TraffikNoise)
{
  TraffiK1 -= TraffikNoise;
  ActualDurationK1 += (TraffiK1 * AutoZeit) + VerlustT;
```

```
}
if (TraffiK3 > TraffikNoise)
{
  TraffiK3 -= TraffikNoise;
  ActualDurationK3 += (TraffiK3 * AutoZeit) + VerlustT;
}
if (TraffiK4 > TraffikNoise)
{
  TraffiK4 -= TraffikNoise;
  ActualDurationK4 += (TraffiK4 * AutoZeit) + VerlustT;
}
if (ActualDurationK1 > MaxTimeK1) ActualDurationK1 = MaxTimeK4;
if (ActualDurationK3 > MaxTimeK3) ActualDurationK3 = MaxTimeK4;
if (ActualDurationK4 > MaxTimeK4) ActualDurationK4 = MaxTimeK4;

ActualDurationK1uK3 = (ActualDurationK1 + ActualDurationK3)/2;
ActualDurationK1 = ActualDurationK1uK3;
ActualDurationK3 = ActualDurationK1uK3;

//Serial.println(TraffiK4);
//Serial.println(TraffiK3);
//Serial.println(TraffiK1);
//Serial.println("");

  // ############################### FR
EnableFR1 = false;
EnableFR3 = false;
EnableFR4 = false;

if((TraffiFR1)){
  //ActualDurationFR1 = MaxFR;
  EnableFR1 = true;
```

```
}
if((TraffiFR3)){
  //ActualDurationFR3 = MaxFR;
  EnableFR3 = true;
}
if((TraffiFR4)){
  //ActualDurationFR4 = MaxFR;
  EnableFR4 = true;
}

TraffiFR1 = false;
TraffiFR3 = false;
TraffiFR4 = false;

  // Rset / Werte löschen
  Loop = 0;
  Relative0 = millis();
  AnimatFR4 = false;
  AnimatFR3 = false;
  AnimatFR1 = false;
  AnimatPfeil = false;
}

//################################## Taster
##############################################
void TasterParser()
{
  // K1
  if (!DetectOUT[0] == 0)
  {
    TraffiK1 += DetectOUT[0];
    DetectOUT[0] = 0;
```

```
}
if (!DetectOUT[6] == 0){
  if(!TraffiK4 == 0) TraffiK1 -= DetectOUT[6];
  DetectOUT[6] = 0;
}
if (!DetectOUT[8] == 0){
  if(!TraffiK4 == 0) TraffiK1 -= DetectOUT[8];
  DetectOUT[8] = 0;
}

// K3
if (!DetectOUT[11] == 0)
{
  TraffiK3 += DetectOUT[11];
  DetectOUT[11] = 0;
}
if (!DetectOUT[12] == 0){
  if(!TraffiK3 == 0) TraffiK3 -= DetectOUT[12];
  DetectOUT[12] = 0;
}
if (!DetectOUT[14] == 0){
  if(!TraffiK3 == 0) TraffiK3 -= DetectOUT[14];
  DetectOUT[14] = 0;
}

// K4
if (!DetectOUT[1] == 0)
{
  TraffiK4 += DetectOUT[1];
  DetectOUT[1] = 0;
}
if (!DetectOUT[5] == 0){
```

```
    if(!TraffiK4 == 0) TraffiK4 -= DetectOUT[5];
    DetectOUT[5] = 0;
  }
}

String TasterParserString()
{
  String out = "";
  out += String(TraffiK1);
  out += String(TraffiK3);
  out += String(TraffiK4);
  return out;
}

// hier noch halt und push detektion hinzufügen
void TasterCheck()
{
  for (int index = 0; index < ELEMENTCOUNT(Detect); index++)
  {
    if (digitalRead(Detect[index]) == 1 && !DetectPressed[index])
    {
      DetectOUT[index]++;
      DetectPressed[index] = true;
    }
    else if (digitalRead(Detect[index]) == 0)
    {
      DetectPressed[index] = false;
    }
  }
}

String TasterCheckString()
```

```
{
 String out = "";
 for (int index = 0; index < ELEMENTCOUNT(Detect); index++)
 {
  out += String(DetectOUT[index]);
 }
 return out;
}

//#################################### FR
########################################
void FRParser()
{
 // FR1
 if (!FROUT[1] == 0 || !FROUT[3] == 0)
 {
  TraffiFR1 = true;
  FROUT[3] = 0;
  FROUT[1] = 0;
 }

 // FR3
 if (!FROUT[8] == 0 || !FROUT[11] == 0)
 {
  TraffiFR3 = true;
  FROUT[11] = 0;
  FROUT[8] = 0;
 }

 // FR4
 if (!FROUT[7] == 0 || !FROUT[4] == 0)
 {
```

```
    TraffiFR4 = true;
    FROUT[7] = 0;
    FROUT[4] = 0;
  }
}

String FRParserString()
{
  String out = "";
  out += String(TraffiFR1);
  out += String(TraffiFR3);
  out += String(TraffiFR4);
  return out;
}

void FRCheck()
{
  for (int index = 0; index < ELEMENTCOUNT(FR); index++)
  {
    if (digitalRead(FR[index]) == 1 && !FRPressed[index])
    {
      FROUT[index]++;
      FRPressed[index] = true;
    }
    else if (digitalRead(FR[index]) == 0)
    {
      FRPressed[index] = false;
    }
  }
}

String FRCheckString()
```

```
{
 String out = "";
 for (int index = 0; index < ELEMENTCOUNT(FR); index++)
 {
  out += String(FROUT[index]);
 }
 return out;
}
```

Interface.cpp

```
#include "interface.hpp"
#include <Arduino.h>

const uint8_t Lampen[16] = {2, 3, 4, 5, 6, 7, 8, 9, A0, A1, A2, A3, A4, A5, A6, A7};
const uint16_t SchaltZeit = 300;

// K4
bool statK4 = true;
bool animatDoneK4 = false;
uint32_t pinStartK4 = 3;
uint32_t goTimeK4 = 0;
uint32_t animatK4 = 0;

// K3
bool statK3 = false;
bool animatDoneK3 = false;
uint32_t pinStartK3 = 0;
uint32_t goTimeK3 = 0;
uint32_t animatK3 = 0;

// K1
bool statK1 = false;
bool animatDoneK1 = false;
```

```
uint32_t pinStartK1 = 9;
uint32_t goTimeK1 = 0;
uint32_t animatK1 = 0;

// Pfeil
bool statPfeil = false;

// FR4
bool statFR4 = false;
bool animatDoneFR4 = false;
uint32_t pinStartFR4 = 14;

// FR3
bool statFR3 = false;
bool animatDoneFR3 = false;
uint32_t pinStartFR3 = 12;

// FR 1
bool statFR1 = false;
bool animatDoneFR1 = false;
uint32_t pinStartFR1 = 6;


// send true to set green
void setK4(bool out)
{
 if (goTimeK4 < millis())
 {
   animatDoneK4 = false;
   if (out)
   {
    if (animatK4 == 0)
```

```
  {
    digitalWrite(Lampen[pinStartK4 + 1], HIGH);
    goTimeK4 = SchaltZeit + millis();
    animatK4++;
  }
  else if (animatK4 == 1)
  {
    digitalWrite(Lampen[pinStartK4 + 2], LOW);
    digitalWrite(Lampen[pinStartK4 + 1], LOW);
    digitalWrite(Lampen[pinStartK4], HIGH);
    statK4 = true;

    animatDoneK4 = true;
    goTimeK4 = 0;
    animatK4 = 0;
  }
}
else
{
  if (animatK4 == 0)
  {
    digitalWrite(Lampen[pinStartK4], LOW);
    digitalWrite(Lampen[pinStartK4 + 1], HIGH);
    goTimeK4 = SchaltZeit + millis();
    animatK4++;
  }
  else if (animatK4 == 1)
  {
    digitalWrite(Lampen[pinStartK4 + 1], LOW);
    digitalWrite(Lampen[pinStartK4 + 2], HIGH);
    statK4 = false;
    animatDoneK4 = true;
```

```
      animatK4 = 0;
      goTimeK4 = 0;
     }
    }
  }
}

void setK3(bool out)
{
  if (goTimeK3 < millis())
  {
    animatDoneK3 = false;
    if (out)
    {
     if (animatK3 == 0)
     {
       digitalWrite(Lampen[pinStartK3 + 1], HIGH);
       goTimeK3 = SchaltZeit + millis();
       animatK3++;
     }
     else if (animatK3 == 1)
     {
       digitalWrite(Lampen[pinStartK3 + 2], LOW);
       digitalWrite(Lampen[pinStartK3 + 1], LOW);
       digitalWrite(Lampen[pinStartK3], HIGH);
       statK3 = true;
       animatDoneK3 = true;
       goTimeK3 = 0;
       animatK3 = 0;
     }
    }
    else
```

```
    {
     if (animatK3 == 0)
     {
       digitalWrite(Lampen[pinStartK3], LOW);
       digitalWrite(Lampen[pinStartK3 + 1], HIGH);
       goTimeK3 = SchaltZeit + millis();
       animatK3++;
     }
     else if (animatK3 == 1)
     {
       digitalWrite(Lampen[pinStartK3 + 1], LOW);
       digitalWrite(Lampen[pinStartK3 + 2], HIGH);
       statK3 = false;
       animatDoneK3 = true;
       animatK3 = 0;
       goTimeK3 = 0;
     }
    }
  }
}

void setK1(bool out)
{
  if (goTimeK1 < millis())
  {
    animatDoneK1 = false;
    if (out)
    {
     if (animatK1 == 0)
     {
       digitalWrite(Lampen[pinStartK1 + 1], HIGH);
       goTimeK1 = SchaltZeit + millis();
```

```
   animatK1++;
  }
  else if (animatK1 == 1)
  {
   digitalWrite(Lampen[pinStartK1 + 2], LOW);
   digitalWrite(Lampen[pinStartK1 + 1], LOW);
   digitalWrite(Lampen[pinStartK1], HIGH);
   statK1 = true;

   animatDoneK1 = true;
   goTimeK1 = 0;
   animatK1 = 0;
  }
 }
 else
 {
  if (animatK1 == 0)
  {
   digitalWrite(Lampen[pinStartK1], LOW);
   digitalWrite(Lampen[pinStartK1 + 1], HIGH);
   goTimeK1 = SchaltZeit + millis();
   animatK1++;
  }
  else if (animatK1 == 1)
  {
   digitalWrite(Lampen[pinStartK1 + 1], LOW);
   digitalWrite(Lampen[pinStartK1 + 2], HIGH);
   statK1 = false;
   animatDoneK1 = true;
   animatK1 = 0;
   goTimeK1 = 0;
  }
```

```
   }
  }
}
 // ##################################### pfeil ######################
void setPfeil(bool inp)
{
  if (inp)
  {
    digitalWrite(Lampen[pinStartK1 -1], HIGH);
    statPfeil = true;
  }
  else
  {
    digitalWrite(Lampen[pinStartK1 -1], LOW);
    statPfeil = false;
  }
}

//############################################ FR
##################################
void setFR4(bool inp)
{
  if (inp)
  {
    digitalWrite(Lampen[pinStartFR4 + 1], LOW);
    digitalWrite(Lampen[pinStartFR4], HIGH);
    statFR4 = true;
    animatDoneFR4 = true;
  }
  else
  {
    digitalWrite(Lampen[pinStartFR4], LOW);
```

```
    digitalWrite(Lampen[pinStartFR4 + 1], HIGH);
    statFR4 = false;
    animatDoneFR4 = true;
  }
}

void setFR3(bool inp)
{
  if (inp)
  {
    digitalWrite(Lampen[pinStartFR3 + 1], LOW);
    digitalWrite(Lampen[pinStartFR3], HIGH);
    statFR3 = true;
    animatDoneFR3 = true;
  }
  else
  {
    digitalWrite(Lampen[pinStartFR3], LOW);
    digitalWrite(Lampen[pinStartFR3 + 1], HIGH);
    statFR3 = false;
    animatDoneFR3 = true;
  }
}

void setFR1(bool inp)
{
  if (inp)
  {
    digitalWrite(Lampen[pinStartFR1 + 1], LOW);
    digitalWrite(Lampen[pinStartFR1], HIGH);
    statFR1 = true;
    animatDoneFR1 = true;
```

```
 }
 else
 {
  digitalWrite(Lampen[pinStartFR1], LOW);
  digitalWrite(Lampen[pinStartFR1 + 1], HIGH);
  statFR1 = false;
  animatDoneFR1 = true;
 }
}

//############################## stat return ##############################
bool getFR3()
{
 return statFR3;
}

bool getFR1()
{
 return statFR1;
}

//############################## Done return ##########################
bool getDoneK4()
{
 return animatDoneK4;
}

bool getDoneK3()
{
 return animatDoneK3;
}
```

Ampel.hpp

```
#pragma once

#include <Arduino.h>

#define ELEMENTCOUNT(x) (sizeof(x) / sizeof(x[0]))

// Taster Analyse und Auswertung
void TasterCheck();
String TasterParserString();
void TasterParser();
String TasterCheckString();

// FR
void FRParser();
String FRParserString();
void FRCheck();
String FRCheckString();

// logic
void TimingSet();
```

interface.hpp

```
#pragma once

#include <inttypes.h>

extern const uint8_t Lampen[16];

void setK4(bool out);
void setK3(bool out);
void setK1(bool out);

void setPfeil(bool inp);
```

```
void setFR4(bool inp);
void setFR3(bool inp);
void setFR1(bool inp);

bool getFR3();
bool getFR1();

bool getDoneK4();
bool getDoneK3();
```

## Abbildungsverzeichnis

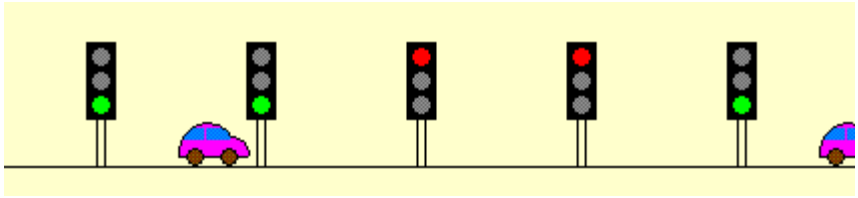



Abbildung 1 Quelle:

Abbildung 2 Quelle: Privat

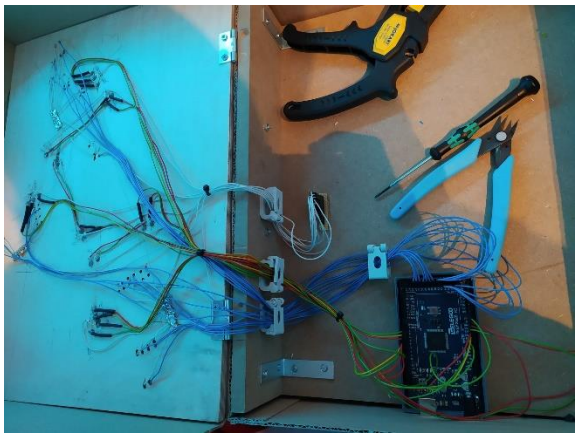


Abbildung 3 Quelle: Privat

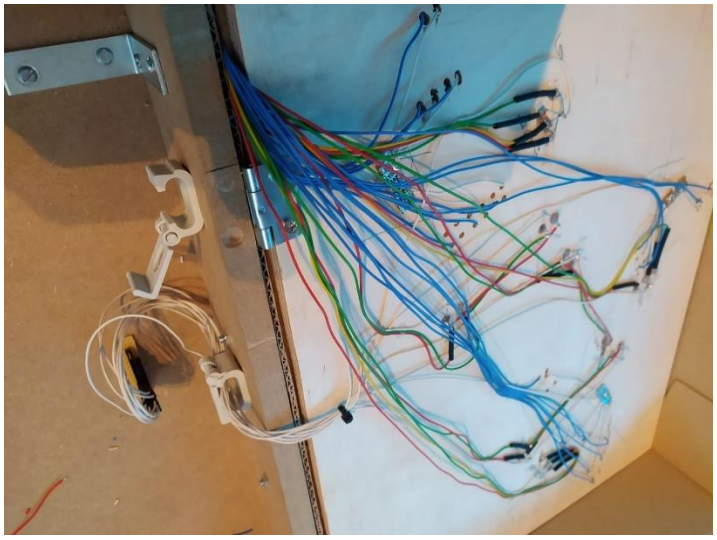


Abbildung 4 Quelle: Privat


Abbildung 5 Quelle: Privat

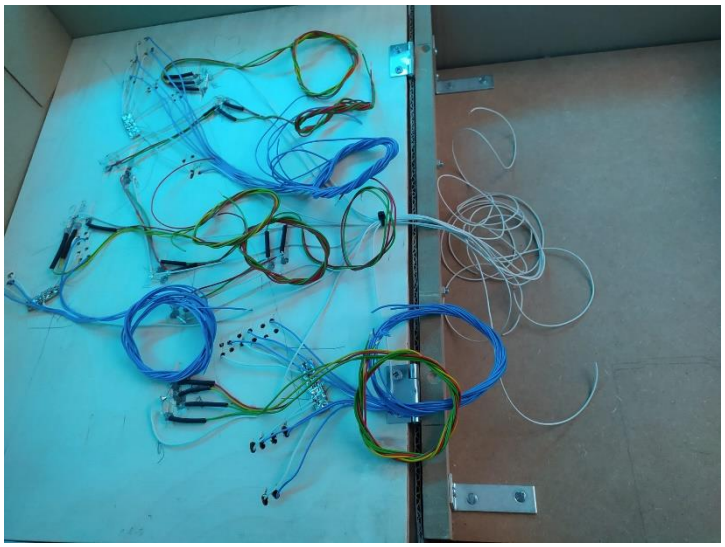


Abbildung 6 Quelle: Privat

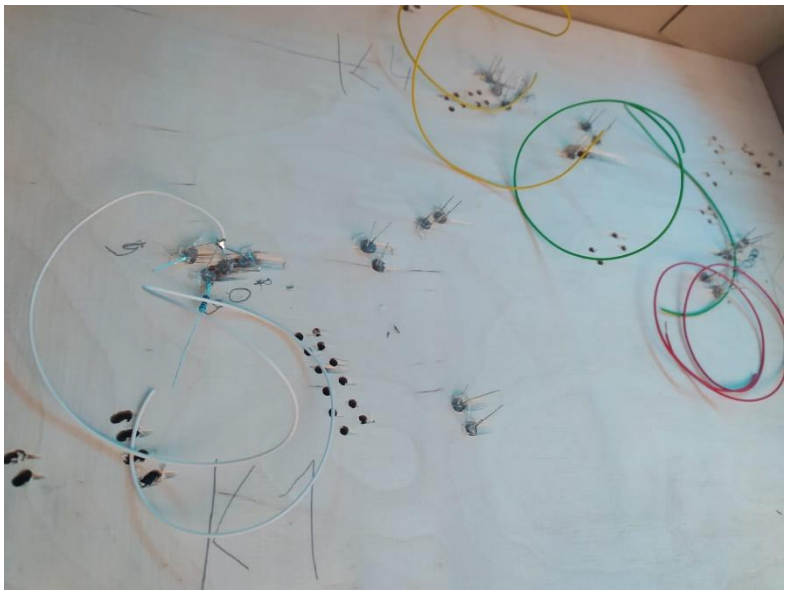


Abbildung 7 Quelle: Privat

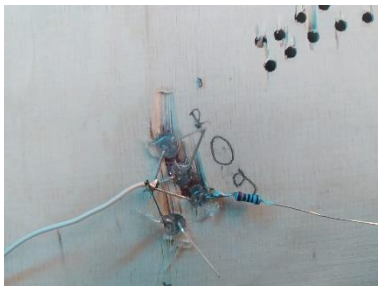


Abbildung 8 Quelle: Privat

Abbildung 9 Quelle: Privat

Abbildung 10 Quelle: Privat

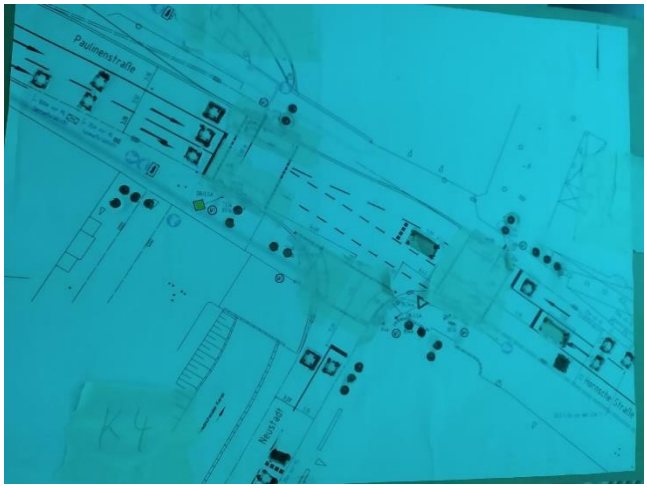


Abbildung 11 Quelle: Privat

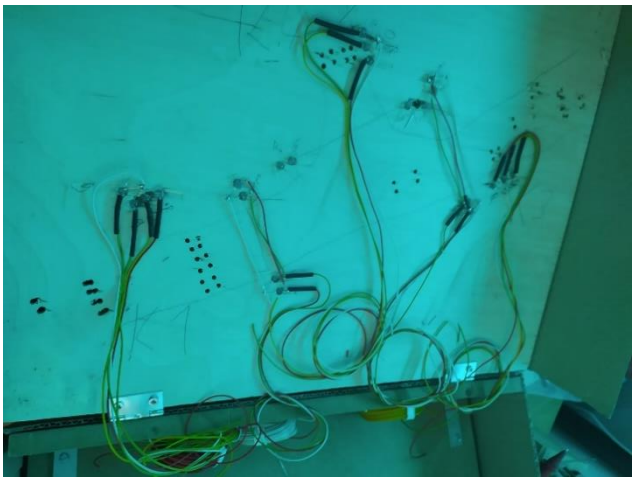
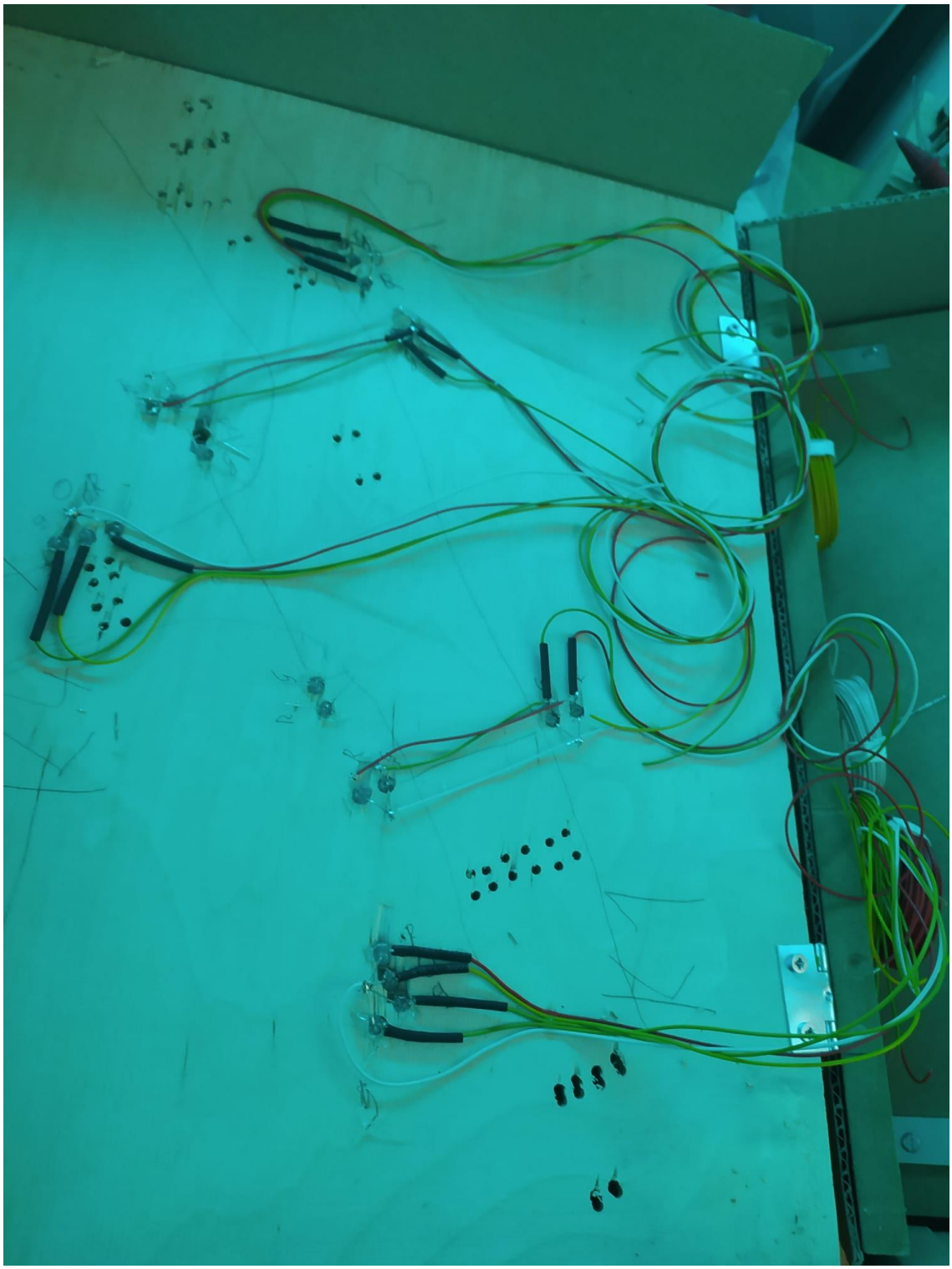
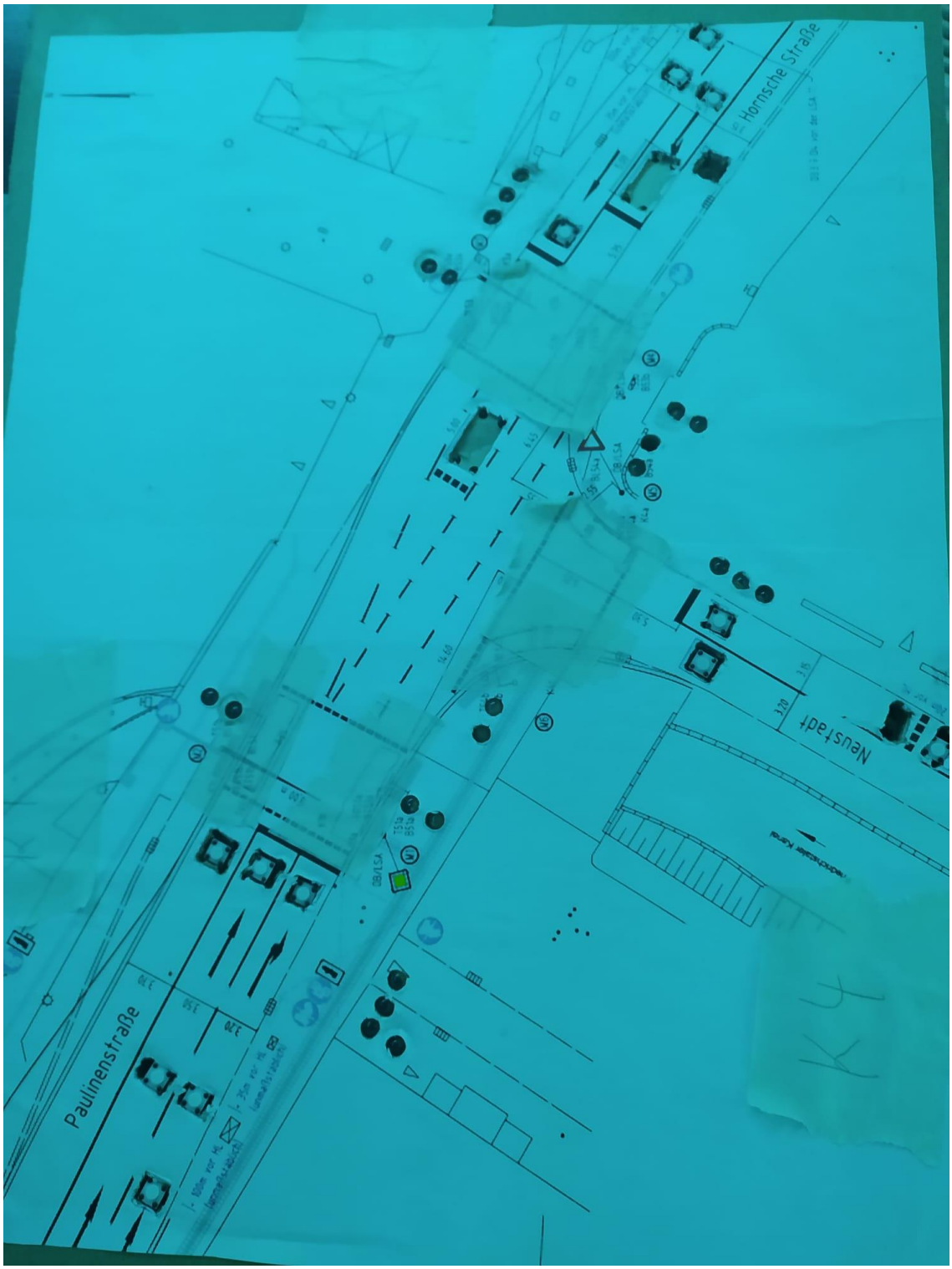


Abbildung 12 Quelle: Privat




Hornsche Straße
Neustadt
Paulinenstraße

Webseiten:

ttps://www.landesverkehrswacht.de/ratgeber/artikel-detail/die-ampel/

## Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die im Literatur– und Quellenverzeichnis angegebenen Hilfsmittel verwendet habe.

Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

(Ort und Datum) (Unterschrift)