GOETHE-GYMNASIUM REGENSBURG

SCIENCE CLUB

LED-Blinker und Ampelsteuerung mit Raspberry Pi

Abschlussbericht

I. Einführung

Das Projekt „LED-Blinker und Ampelsteuerung“ diente dazu, Schülerinnen und Schülern eine praxisnahe Einführung in Programmierung und Hardwarekomponenten zu geben. Mithilfe des Raspberry Pi und grundlegender Elektronikkenntnisse lernten die Teilnehmer, wie sie durch Softwaresteuerung einfache und komplexe Schaltungen umsetzen können.
Das Projekt wurde erfolgreich durchgeführt und ermöglichte den Schülerinnen und Schülern nicht nur technisches Wissen, sondern auch Kreativität, Problemlösungskompetenz und Teamarbeit.

II. Ziele des Projekts

1. **Praktisches Verständnis von Programmierung und Hardware**
   Die Schüler erlernten die Grundlagen der Programmiersprachen **Scratch** und **Python**, um einfache Schaltungen wie LED-Blinker zu programmieren.

2. **Vertiefung von Elektronikkenntnissen**
   a. Aufbau und Erweiterung von Schaltungen mit LEDs, Widerständen und Breadboards.
   b. Simulation und Steuerung einer realistischen Ampelschaltung mit mehreren LEDs.
3. **Förderung logischen Denkens und Problemlösungsstrategien**
   Die Schüler entwickelten und optimierten eigenständig Programme, um die verschiedenen Zustände der Ampel zu steuern und Synchronisationsprobleme zu lösen.

4. **Abschlusspräsentation und Reflexion**
   Am jährlichen Schulweiten MINT-Projekt-Tag präsentierten die Schüler der einzelnen Teams ihre Ergebnisse und reflektierten über ihre Lernfortschritte und Herausforderungen.

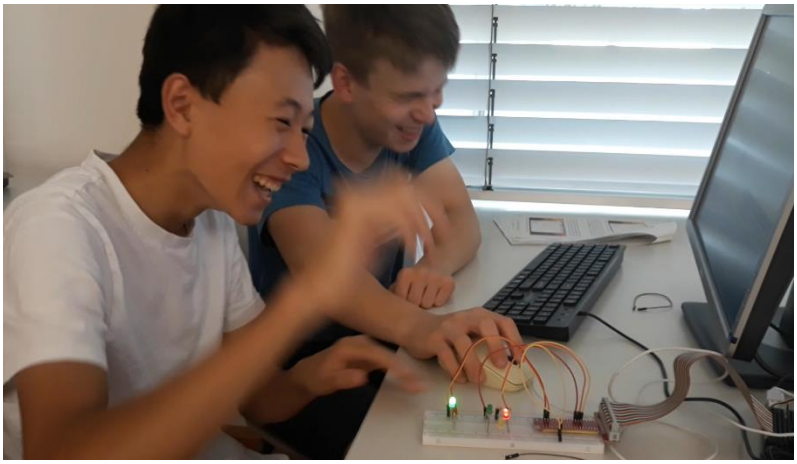
III. Projektverlauf

1. **Grundlagen**

   - Einführung in die Nutzung des **Raspberry Pi** und dessen GPIO-Ports.
   - Erste Erfahrungen mit Scratch und Python: Erstellung eines einfachen Blinkprogramms, um LEDs anzusteuern.
   - Aufbau der Schaltungen auf Breadboards mit LEDs und Widerständen.

2. **LED-Blinker**

   - Programmierung eines einfachen LED-Blinkers in Scratch und Python.
   - Vergleich der beiden Programmiersprachen hinsichtlich Benutzerfreundlichkeit und Einsatzmöglichkeiten.



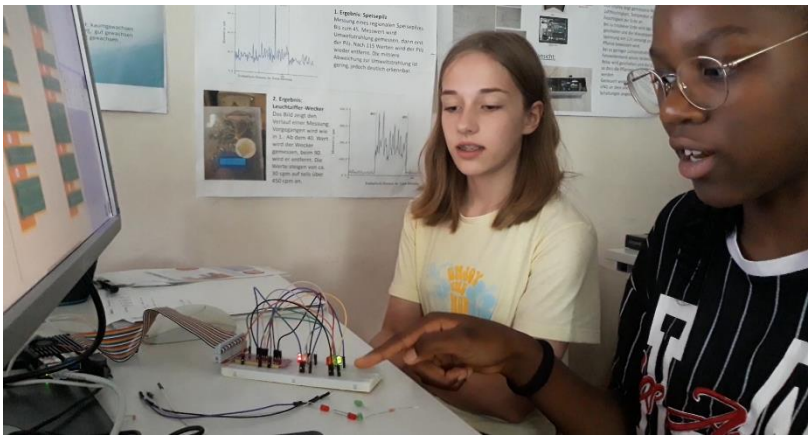
3. **Fußgängerampel**

   - Erweiterung der Schaltung zur Simulation einer Fußgängerampel mit mehreren LEDs.
   - Implementierung der Steuerungslogik in Scratch und Python: Grün, Gelb, Rot für Fahrzeuge und Grün, Rot für Fußgänger.
   - Verwendung von Kontrollstrukturen wie `if-else` und Schleifen zur Steuerung der Übergänge zwischen den Ampelphasen.
   - Tests und Fehlerbehebungen zur Sicherstellung der Funktionalität und Sicherheit.

4. **Abschlusspräsentation**

   - Am MINT-Projekt-Tag im Februar präsentierten die Schüler ihre Projekte und erläuterten die Funktionsweise ihrer Programme.
   - Sie stellten Herausforderungen und Verbesserungsmöglichkeiten vor, z. B. Optimierungen zur Messung und Steuerung des „Verkehrsflusses“ in der Simulation.

IV. Ergebnisse und Erfolge



1. **Technische Kompetenz:**
   Die Schüler beherrschten am Ende des Projekts grundlegende Programmierkonzepte sowie den Umgang mit elektronischen Komponenten und Schaltlogik.
2. **Kreativität und Problemlösung:**
   Jedes Team entwickelte individuelle Ansätze zur Optimierung ihrer Ampelsteuerungen. Einige Teams fügten zusätzliche Funktionen hinzu, wie variable Wartezeiten für Fußgänger und selbstfahrende Fahrzeuge.
3. **Teamarbeit:**
   In kleinen Gruppen arbeiteten die Schüler effizient zusammen, halfen sich gegenseitig bei Problemen und entwickelten innovative Lösungen.
4. **Reflexion und Präsentationsfähigkeit:**
   Durch die Abschlusspräsentation lernten die Schüler, ihre Arbeit verständlich und selbstbewusst zu präsentieren, was ihre Kommunikationsfähigkeit stärkte.

V. Fazit und Ausblick

Das Projekt war ein voller Erfolg. Es ermöglichte den Schülerinnen und Schülern, theoretische Konzepte durch praktische Anwendungen zu verstehen. Die Kombination aus Elektronik und Programmierung machte das Lernen spannend und förderte die Begeisterung für Elektronik und Technik.

Zukünftige Projekte: Mit der Förderung durch den Wettbewerb **"LABS for CHIPS"** wird das Projekt im kommenden Schuljahr erneut durchgeführt, um weiterhin junge Menschen für Mikroelektronik und ihre vielfältigen Anwendungen zu gewinnen. Dies bietet uns die Möglichkeit, das Konzept weiter auszubauen und noch mehr Schüler mit den spannenden Möglichkeiten moderner Technologien zu fördern. Aufbauend auf diesem Projekt könnten komplexere Themen wie die Integration von Sensoren oder die Steuerung per App umgesetzt werden, um die Schüler weiter zu fordern und ihre Kompetenzen auszubauen.

Auszüge aus den Schülerarbeiten

Lösungsansatz mit der Programmierumgebung SCRATCH: Die Nutzung des Raspberry Pi mit Scratch erlaubt eine schrittweise Annäherung von einfachen LED-Blink-Programmen bis hin zu komplexeren Ampellogiken ganz ohne große Programmierkenntnisse.

Einleitung

Da wir seit dem Schuljahr 2023/24 beim Wahlfach ScienceClub mitmachten, wurde uns dort dieses Projekt vorgeschlagen. Obwohl wir erst in der 7. Jahrgangsstufe waren konnten wir alles verstehen und umsetzen. Es wurde von uns, Sydney und Matilda, bearbeitet. Dabei wurden zwei Ampeln auf Scratch so programmiert, dass sie funktionieren wie als wären sie in einer Kreuzung eingebaut.

Projektaufgabe

1. **Kennenlernen der technischen Bedingungen:** Aufbau und Steuerung einer Ampel mit LEDs unter Nutzung der GPIO-Pins des Raspberry Pi; Einführung in die Nutzung von Komponenten wie Widerständen, Breadboards und LEDs.
2. **Erarbeiten der Programme:** Grundlagen in Scratch insbesondere der Arbeit mit Kontrollstrukturen.
3. **Bau einer realistischen Ampelumgebung; Kreativität:** Eigenständige Fehlersuche, Optimierung der Schaltungen und Reflexion über Erweiterungsmöglichkeiten.

Projektaufbau und Material

Komponentenliste:

Komponente	Details
LEDs (Rot, Gelb, Grün)	Ø 5 mm
Widerstände	470 Ω und 10k Ω
Raspberry Pi	Raspberry Pi 5 mit 8 GB RAM
Breadboard	Standard Experimentier-Steckboard
Verbindungskabel	Jumper-Kabel (Dupont)

Zusätzlich wurde Raspberry Pi OS installiert, das die nötigen Softwarepakete (z.B. GPIO-Bibliotheken für Python) bereitstellt

Technischer Aufbau

1. **GPIO-Erweiterung aktivieren:**
   - Öffne Scratch.
   - Gehe zu „Erweiterungen hinzufügen“ und wähle **GPIO** aus.
2. **Ampel-Logik programmieren:**

Beispielcode in SCRATCH:

```
"Wenn grüne Flagge angeklickt"
Wiederhole fortlaufend:
    - Schalte die grüne LED an: "Setze GPIO 22 auf [Hoch]"
    - Warte 5 Sekunden
    - Schalte die grüne LED aus und die gelbe LED an: "Setze GPIO 22 auf
[Niedrig]", "Setze GPIO 27 auf [Hoch]"
    - Warte 2 Sekunden
    - Schalte die gelbe LED aus und die rote LED an: "Setze GPIO 27 auf
[Niedrig]", "Setze GPIO 17 auf [Hoch]"
    - Warte 5 Sekunden

    - Schalte die rote LED aus: "Setze GPIO 17 auf [Niedrig]"
```

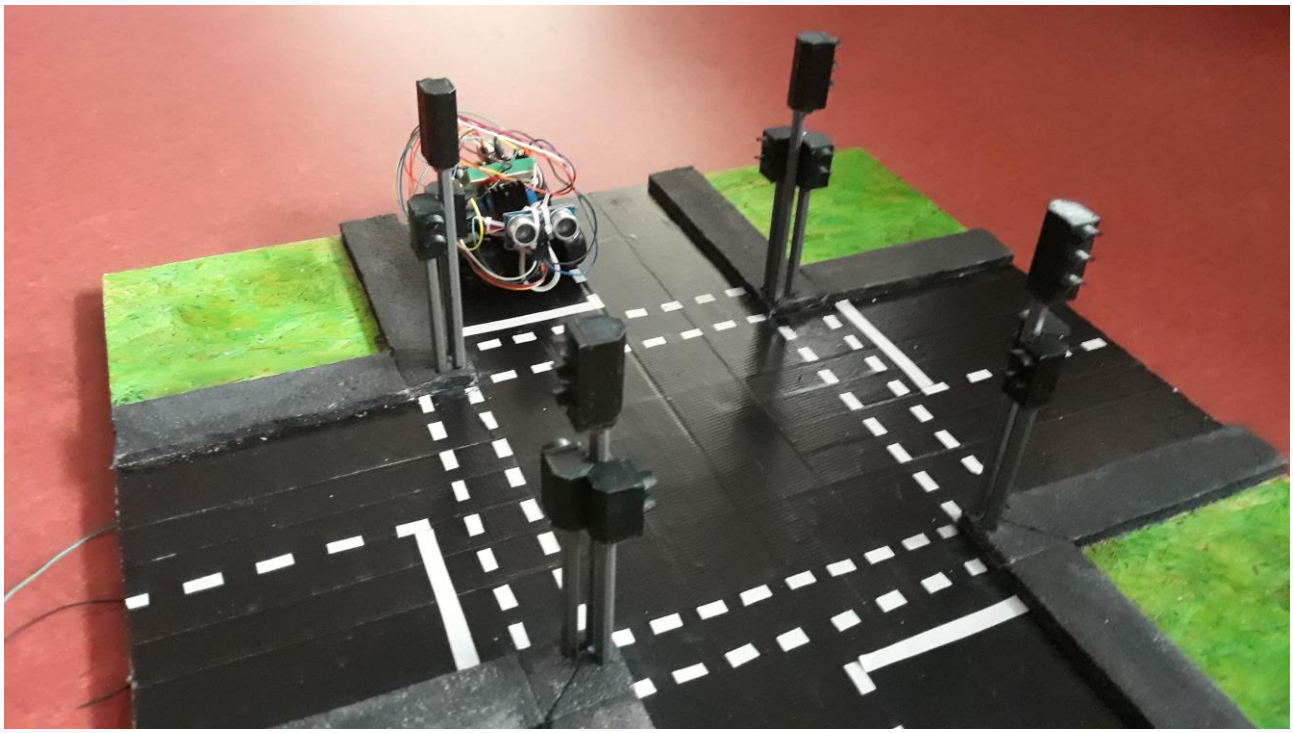
Die GPIO-Pins werden in Scratch über die GPIO-Blöcke (z. B. „Setze GPIO 22 auf [Hoch]“) angesteuert.

Eine Schleife sorgt dafür, dass die Ampel dauerhaft durch die Phasen wechselt.

Erweiterungen:

Fußgängerampel: Füge zusätzliche GPIO-Pins für Fußgänger-LEDs hinzu und programmiere die Phasen synchron zur Fahrzeugampel.

Tastersteuerung: Verbinde einen Taster mit einem GPIO-Pin, um das Fußgängersignal manuell zu aktivieren. Dies kann mit einer Bedingung (Wenn GPIO X gedrückt) in Scratch umgesetzt werden.



Lösungsansatz mit der Programmierung unter PYTHON: Die Nutzung des Raspberry Pi mit Python erlaubt eine tiefergehende und komplexeren Ampellogik zu erstellen. Die nötigen Programmierkenntnisse wurden im Laufe dieses Projektes erlernt.

Einleitung

Lisa und Jonathan entwickelten im Rahmen eines Projektes im ScienceClub (offene Schülerwerkstatt des Goethe-Gymnasiums) ein Ampelkreuzungsmodell mit einem Raspberry Pi 5. Das Projekt kombiniert Hardware- und Software-Komponenten und stellt ein funktionierendes Verkehrsmodell dar. Ziel war es, eine programmierte Verkehrssteuerung zu erstellen und ein Bluetooth-gesteuertes Auto zu integrieren. Dieses Projekt soll zeigen, wie Schüler ab 12 Jahren mit grundlegenden Programmier- und Elektronikkenntnissen ein solches Modell realisieren können.

Das Auto, das wir gebaut und bei grün über die Kreuzung fahren lassen haben

Materialbedarf:

- Raspberry Pi 5
- Evtl. GPIO-Erweiterung für RP5 und Breadboard
- LEDs in Rot, Gelb und Grün; je nach Art und Anzahl der Ampeln
  Bei uns: 12 x rot, 4 x gelb, 12 x grün (4 Verkehrsampeln, 8 Fußgängerampeln)
- Kabel in grün, gelb, rot, schwarz
  Bei uns: 12 x rot, 4 x gelb, 12 x grün, 12 x schwarz/weiß, je 50 cm lang
- Widerstände (330 Ohm)
  Bei uns: 28 Widerstände
- Zugang zu einem 3D-Drucker
- (Sperr-) Holz für Untergrund/Modellbasis, Scharniere und Winkel
- Bei uns: 2 x 60x60 cm, 4 x 10x60 cm Sperrholzplatten, 1 Scharnier, 3 Winkel
- Tape in schwarz und weiß
- Acrylfarben (grün, grau, schwarz)

- (Arduino-basiertes programmierbares Auto mit Bluetooth Low Power, passende Größe für das Modell)

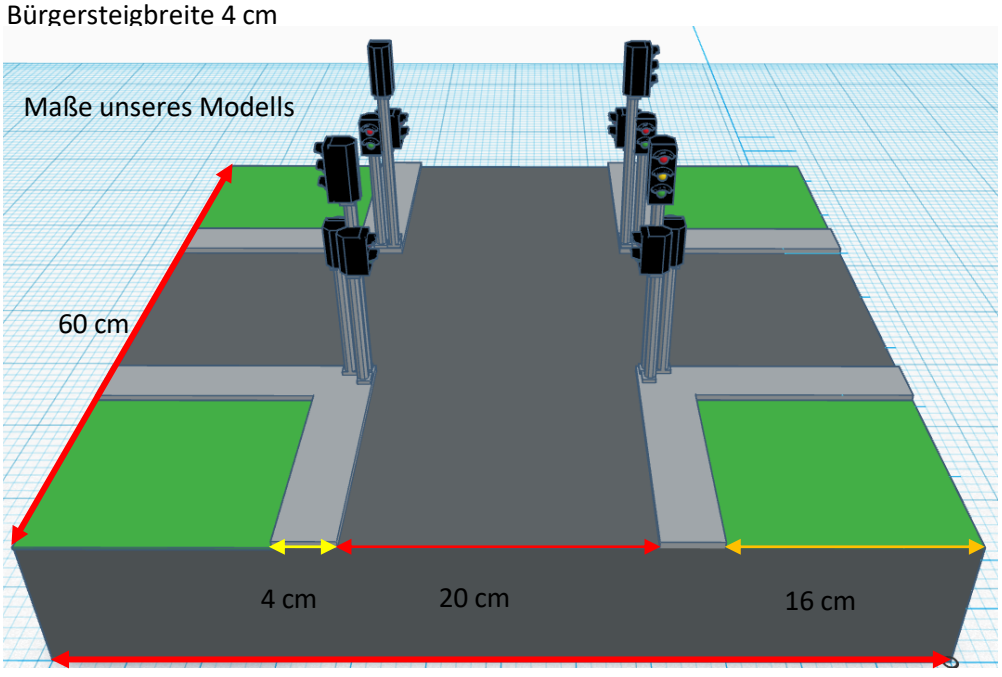
Geräteliste:

- Bosch Stichsäge PST 650
- Lötkolben und Lötzinn
- Dremel 3000 Multi-Function Tool
- Bosch Home and Garden 06039D4109 Bosch Hammer Drill
- ggf. Säge
- Schere oder Bastelmesser
- Kabelschere (o. Ä. zum Abisolieren der Kabel)

Maße unseres Modells

ANLEITUNG

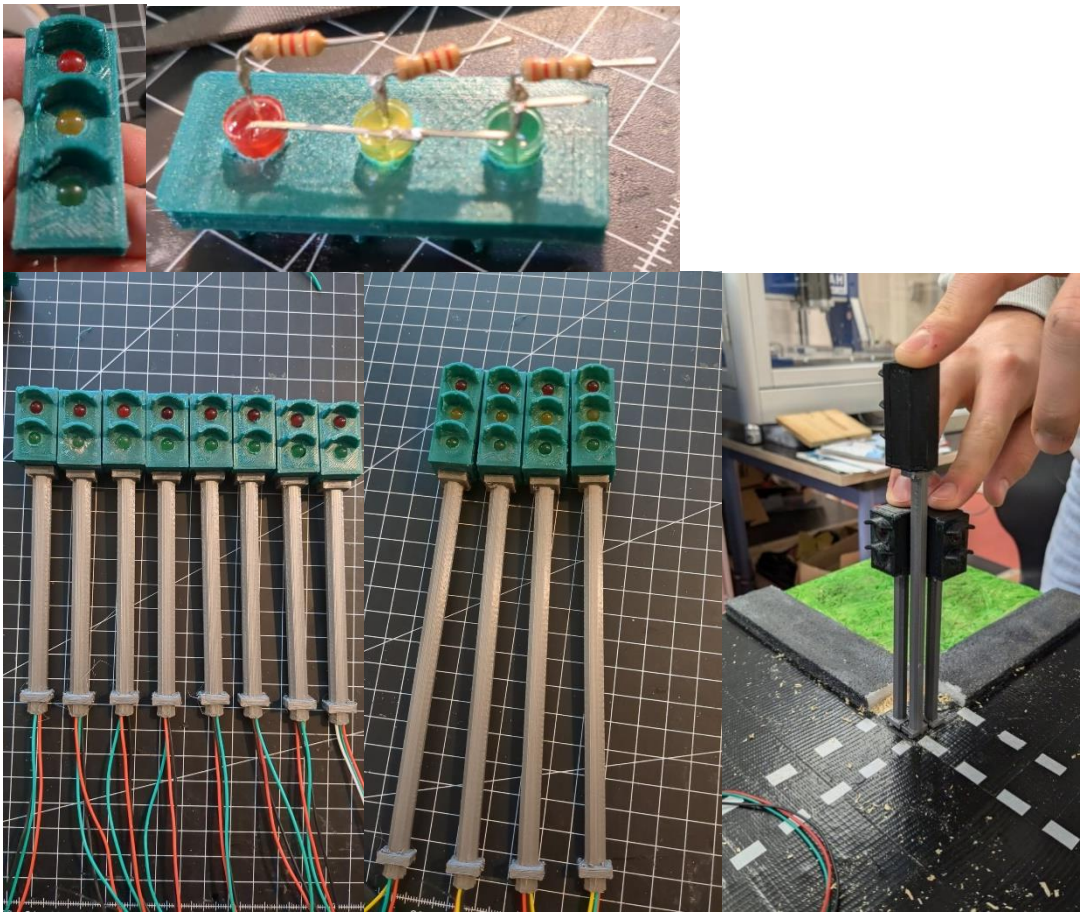
1. Beschaffung der Materialien
2. Raspberry Pi 5 einrichten und Code übertragen (entweder nur den Ampelcode oder den mit zusätzlicher Autosteuerung, allerdings können Anpassungen erforderlich sein, je nach Hardware und Steuerungsart)
3. Modell bauen:
    1) Säge die Holzplatten zurecht, bei uns: 2x 60x60 cm für Bodenplatte und Modellbasis, 4x 60x10 cm für Seitenplatten
    2) Recherchiere nach den realen Maßen für Straßen, Ampeln und Bürgersteige und rechne sie auf den gewünschten Maßstab um bei uns: Straßenbreite 20 cm, Spurbreite 10 cm, Bürgersteigbreite 4 cm



3) Drucke die Ampeln

4) Ampeln zusammenbauen:
In die Löcher der Ampel-Deckplatte werden die LEDs in der Reihenfolge rot, gelb, grün (von oben nach unten) so gesteckt (wenn die LEDs nicht reinpassen, Löcher größer feilen) , dass alle GND-Beine auf einer und alle VCC-Beine jeweils auf der anderen Seite der Rückseite sind.
Anschließend werden alle GND-Beine zusammengebogen, evtl. auf die richtige Länge zugeschnitten und zusammengelötet.
Löte das schwarze Kabel an das Ende der GND-Seite und kürze die VCC-Beine so, dass sie in das Gehäuse passen, wenn die Deckplatte aufliegt. Löte jeweils die Widerstände an die gekürzten Beine der LEDs.

Löte die der LED-Farbe entsprechend gefärbten Kabel an die Widerstände und biege die Beine so, dass alles noch ins Gehäuse passt.
Führe alle Kabel durch das Loch unten am Gehäuse der Ampel und klebe die Deckplatte auf das Gehäuse (z.B. mit Sekundenkleber).
Führe die Kabel durch den Pfosten der Ampel und Klebe ihn unten an das Gehäuse.
Wiederhole das für alle restlichen Ampeln (für Fußgängerampeln kein gelb)



Löten und Zusammenbau der Ampeln

5) Klebe das schwarze Tape in der gewünschten Straßenbreite auf die Modellbasis (siehe 2) oben) und schneide das weiße Tape in Streifen für die Straßenmarkierungen. Klebe diese in die Mitte der Straße. Vergiss die Markierungen für den Fußgängerüberweg und den Stoppstreifen für Autos bei der Ampel nicht! Orientiere dich dabei an der realen Welt.

6) Baue den Bürgersteig (4cm breit) aus einem geeigneten Material, male ihn grau an und klebe ihn auf die Modellplatte. Male den Grasbereich grün an. Bohre anschließend mit einem Dremel (o.Ä. wie Bohrer) ca. 8 mm große Löcher an die Ecken des Bürgersteigs, stecke die Kabel der Ampeln hindurch und befestige diese mit z.B. Sekundenkleber (siehe 3d-Modell oben bei 2).

7) Überspringe diesen Schritt, wenn du nur die Ampelkreuzung, ohne Auto bauen willst. Lade den untenstehenden BLE-Code für den RPI5 und das Arduino-Bluetooth-Auto jeweils hoch und tausche ggf. den Code für die Autoansteuerung aus. Finde die Service UUID deines Mikrocontrollers z.B. mithilfe der App „nRF Connect“ heraus und tausche sowohl diese, als auch die beiden direkt darunter aus. Starte den Code und schau ob es funktioniert. Wenn nicht erkundige dich im Internet nach Raspberry Pi und Arduino Bluetooth Kommunikation.

8) Verbinde jeweils die gleichfarbigen Kabel der Verkehrsampeln der Längsstraße miteinander und verbinde sie mit dem passenden GPIO-Pin des RPI5. Mache das gleiche für die Querseite und die Fußgängerampeln. Achte darauf, dass die Fußgänger- und Verkehrsampeln unterschiedlich angesteuert werden, also nur die gleiche Ampelart miteinander verbunden wird.

9) Bringe die Holzseiten mithilfe von Winkeln und Scharnieren an der Modellbasis an (male die Seiten ggf. an). Überlege dir ein gutes Kabelmanagement und stelle die Modellbasis auf die Unterplatte.

CODE

Nur Ampeln

```
from gpiozero import LED
from time import sleep

led_gr = LED(13)
led_ge = LED(19)
led_r = LED(26)

led2_gr = LED(27)
led2_ge = LED(22)
led2_r = LED(5)

while True:
    led_gr.on()

    sleep(5)

    led_gr.off()
    led_ge.on()

    sleep(2)

    led_ge.off()
    led_r.on()

    sleep(2)

    sleep(15)

    sleep(2)

    led_ge.on()

    sleep(2)

    led_ge.off()
    led_r.off()
    led_gr.on()

    sleep(5)
```

Ampeln mit Auto und Bluetooth

Python für Raspberry Pi 5

```
import sys
import time

from gpiozero import LED

r_v = LED(17)
y_v = LED(27)
g_v = LED(22)

r_h = LED(13)
y_h = LED(19)
g_h = LED(26)

g_d = 15
y_d = 3
r_d = 3
ry_d = 2

import requests
from PyQt5.QtCore import QObject, QRunnable, QThreadPool, QTimer, pyqtSignal,
pyqtSlot
from bluepy import btle

class WorkerSignals(QObject):
    signalMsg = pyqtSignal(str)
    signalRes = pyqtSignal(str)

class MyDelegate(btle.DefaultDelegate):

    def __init__(self, sgn):
        btle.DefaultDelegate.__init__(self)
        self.sgn = sgn

    def handleNotification(self, cHandle, data):

        try:
            dataDecoded = data.decode()
            self.sgn.signalRes.emit(dataDecoded)
        except UnicodeError:
            print("UnicodeError: ", data)

class WorkerBLE(QRunnable):

    def __init__(self):
        super().__init__()
        self.signals = WorkerSignals()
        self.rqsToSend = False

    @pyqtSlot()
    def run(self):
        #-----------------------------------------------
        p = btle.Peripheral("ec:da:3b:37:36:5e")
        p.setDelegate( MyDelegate(self.signals) )

        svc = p.getServiceByUUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
        self.ch_Tx = svc.getCharacteristics("6E400002-B5A3-F393-E0A9-
E50E24DCCA9E")[0]
```

```
        ch_Rx = svc.getCharacteristics("6E400003-B5A3-F393-E0A9-
E50E24DCCA9E")[0]

        setup_data = b"\x01\00"
        p.writeCharacteristic(ch_Rx.valHandle+1, setup_data)

        # BLE loop --------

        while True:
            """
            if p.waitForNotifications(1.0):
                # handleNotification() was called
                continue

            print("Waiting...")
            """

            p.waitForNotifications(0.5)

            if self.rqsToSend:
                self.rqsToSend = False

                try:
                    self.ch_Tx.write(self.bytestosend, True)
                except btle.BTLEException:
                    print("btle.BTLEException");

    def toSendBLE(self, tosend):
        self.bytestosend = bytes(tosend, 'utf-8')
        self.rqsToSend = True
        """
        try:
            self.ch_Tx.write(bytestosend, True)
        except BTLEException:
            print("BTLEException");
        """

class RunCommands():

    def __init__(self):
        super().__init__()
        self.threadpool = QThreadPool()
        self.startBLE()

    def startBLE(self):
        self.workerBLE = WorkerBLE()
        self.workerBLE.signals.signalMsg.connect(self.slotMsg)
        self.workerBLE.signals.signalRes.connect(self.slotRes)
        self.threadpool.start(self.workerBLE)

    def sendBLE(self, strToSend):
        self.workerBLE.toSendBLE(strToSend)

    def slotMsg(self, msg):
        print(msg)

    def slotRes(self, res):
        self.console.appendPlainText(res)

# main stuff
```

```
c = RunCommands()
while True:

    c.sendBLE("start")
    r_v.off()
    y_v.off()
    g_v.on()
    time.sleep(g_d)

    #c.sendBLE("|yellow")
    g_v.off()
    y_v.on()
    time.sleep(y_d)

    c.sendBLE("|red")
    y_v.off()
    r_v.on()
    time.sleep(r_d)

    #c.sendBLE("-red + yellow")
    y_h.on()
    time.sleep(ry_d)

    c.sendBLE("-green")
    r_h.off()
    y_h.off()
    g_h.on()
    time.sleep(g_d)

    #c.sendBLE("-yellow")
    g_h.off()
    y_h.on()
    time.sleep(y_d)

    c.sendBLE("-red")

    y_h.off()
    r_h.on()
    time.sleep(r_d)

    #c.sendBLE("|red + yellow")
    y_v.on()

    c.sendBLE("stopp")

    time.sleep(ry_d)
```

Arduino Code für Auto

```
/*
    Video: https://www.youtube.com/watch?v=oCMOYS71NIU
    Based on Neil Kolban example for IDF: https://github.com/nkolban/esp32-snippets/blob/master/cpp_utils/tests/BLE%20Tests/SampleNotify.cpp
    Ported to Arduino ESP32 by Evandro Copercini

   Create a BLE server that, once we receive a connection, will send periodic notifications.
   The service advertises itself as: 6E400001-B5A3-F393-E0A9-E50E24DCCA9E
   Has a characteristic of: 6E400002-B5A3-F393-E0A9-E50E24DCCA9E - used for receiving data with "WRITE"
   Has a characteristic of: 6E400003-B5A3-F393-E0A9-E50E24DCCA9E - used to send data with  "NOTIFY"

   The design of creating the BLE server is:
   1. Create a BLE Server
   2. Create a BLE Service
   3. Create a BLE Characteristic on the Service
   4. Create a BLE Descriptor on the characteristic
   5. Start the service.
   6. Start advertising.

   In this example rxValue is the data received (only accessible inside that function).
   And txValue is the data to be sent, in this example just a byte incremented every second.
*/
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

BLEServer *pServer = NULL;
BLECharacteristic *pTxCharacteristic;
bool deviceConnected = false;
bool oldDeviceConnected = false;
uint8_t txValue = 0;

String text;

unsigned long intervall= 2000;

unsigned long before = 0;

// See the following for generating UUIDs:
// https://www.uuidgenerator.net/

#define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"  // UART service UUID
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

class MyServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer *pServer) {
    deviceConnected = true;
  };

  void onDisconnect(BLEServer *pServer) {
    deviceConnected = false;
```

```
  }
};

class MyCallbacks : public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    String rxValue = pCharacteristic->getValue();

    if (rxValue.length() > 0) {
      Serial.println("*********");
      Serial.print("Received Value: ");
      for (int i = 0; i < rxValue.length(); i++) {
        Serial.print(rxValue[i]);
      }
      text=rxValue;

      Serial.println();
      Serial.println("*********");
    }
  }
};

void setup() {
  Serial.begin(115200);

  // Create the BLE Device
  BLEDevice::init("UART Service");

  // Create the BLE Server
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  // Create the BLE Service
  BLEService *pService = pServer->createService(SERVICE_UUID);

  // Create a BLE Characteristic
  pTxCharacteristic = pService->createCharacteristic(CHARACTERISTIC_UUID_TX,
BLECharacteristic::PROPERTY_NOTIFY);

  pTxCharacteristic->addDescriptor(new BLE2902());

  BLECharacteristic *pRxCharacteristic = pService-
>createCharacteristic(CHARACTERISTIC_UUID_RX,
BLECharacteristic::PROPERTY_WRITE);

  pRxCharacteristic->setCallbacks(new MyCallbacks());

  // Start the service
  pService->start();

  // Start advertising
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
}

void loop() {

  if (deviceConnected) {
    pTxCharacteristic->setValue(&txValue, 1);
    pTxCharacteristic->notify();
    txValue++;
```

```
    delay(100); // bluetooth stack will go into congestion, if too many packets
are sent
  }

  // disconnecting
  if (!deviceConnected && oldDeviceConnected) {
    delay(500);                   // give the bluetooth stack the chance to get
things ready
    pServer->startAdvertising();  // restart advertising
    Serial.println("start advertising");
    oldDeviceConnected = deviceConnected;
  }
  // connecting
  if (deviceConnected && !oldDeviceConnected) {
    // do stuff here on connecting
    oldDeviceConnected = deviceConnected;
  }

  if(text=="-green"){
//    Serial.println("Motoren an----------------------");
//    unsigned long now = millis();
//    unsigned long differenz = millis() - before;
    if(millis() - before > intervall){
      before = millis();
      Serial.println("an");
    }
//    Serial.println("aus");

  }
}
```