Schachroboter Doku

Zugehörige Schüler: Matthias Kurz, Ivan Milev, Jakob Scarlata, Christof Zlabinger

Klasse: 5DHIT

1. Beschreibung des Projektes

Unser Ziel ist es, einen Schachroboter zu bauen, der mit jemandem live mit echten physischen Figuren Schachspielen kann.

2. Unsere Idee und Vision

Unser Team, bestehend aus Matthias Kurz, Ivan Milev, Jakob Scarlata und Christof Zlabinger, verfolgt die Vision, innovative Technik mit spielerischer Herausforderung zu verbinden. Im Rahmen des Fachs "Industrielle Informationstechnik" wollen wir einen Roboter entwickeln und programmieren, der Schach spielen kann.

Die zentrale Idee ist, ein Schachbrett aus dünnem Karton auf den Roboter zu legen, wobei die Figuren mit kleinen Magneten ausgestattet werden. Der Roboter soll in der Lage sein, sich präzise in alle vier Richtungen zu bewegen und die Figuren von unten über Magnetismus zu steuern. Damit kombinieren wir Grundlagen der Robotik, Mechanik und Programmierung zu einem kreativen und lehrreichen Projekt, das sowohl technische als auch strategische Anforderungen vereint.

Außerdem soll es möglich sein, über ein externes Modul Schachzüge angeben zu können, wobei der Roboter zieht. Zuzüglich sollen auch Überprüfungen auf legal Züge und eine Benachrichtigung bei einem Schach/ Matt stattfinden.

Im Sommersemester wollen wir zusätzlich auch noch die Option hinzufügen, gegen den Computer zu spielen. Dazu wird ein Schachcomputer, wie Stockfish, hinzufügt, der dann anstelle des zweiten Spielers automatisch die Züge macht.

Logischerweise kann man dann auch die Schwierigkeit angeben, um das Vergnügen, sowie die Herausforderung zu maximieren.

3. Ziele (Wintersemester)

3.1 Das zeigen wir für GKÜ

Es ist eine Kommunikation mit dem Festo über das Canbus Modul möglich und der Roboter kann beliebig angesteuert werden.

3.2 Das zeigen wir für GKV

Unsere Software hat für jede Figur einen Punkt abgespeichert und der Roboter kann auf eines dieser beliebigen Felder fahren

3.3 Wir fühlen uns adventurous und zeigen das für EKÜ

Die Schachzüge sind nun zur einfacheren Darstellung auch in der korrekten Schachnotation angebbar

3.4 Maximum Flex! Das zeigen wir für EKV

Es ist möglich, dass man Schachzüge über ein externes Modul angeben kann

4. Verwendete Materialien

- Roboter: Festo
- Schachbrett: dünnen Karton
- Schachfiguren aus einem normalen Schachbrettset
- Module: MS5Stack mit Drehrad für Steuerung, Canbus Modul für Kommunikation
- Magneten
- Klebeband

- Raspberry Pi
- CAN MCP2515

5. Umsetzung und Durchführung

5.1 Grober Plan und erste Überlegungen

Zuerst haben wir uns in zwei Zweiergruppen aufgeteilt, um eine maximale Effizienz zu gewährleisten.

Gruppe A, bestehend aus Christof und Matthias, setzte sich das Ziel, die Kommunikation zwischen dem Roboter und dem Raspberry Pi erfolgreich umzusetzen. Zudem arbeiteten sie daran, das Mapping der einzelnen Felder des Schachbretts zu realisieren.

Gruppe B, bestehend aus Ivan und Jakob, widmete sich der Herausforderung, den Gegenmagneten unterhalb des Schachbretts anzubringen und wieder zu lösen. Dieser Schritt ist essenziell, um später die Figurenmagneten anzuheben und loszulassen, beispielsweise für das „Schlagen“ von Figuren. Darüber hinaus war Gruppe B auch für die Anfertigung des Schachbretts verantwortlich.

5.2 Schwierigkeiten und erste Probleme (erster Anlauf)

5.2.1 Kommunikation

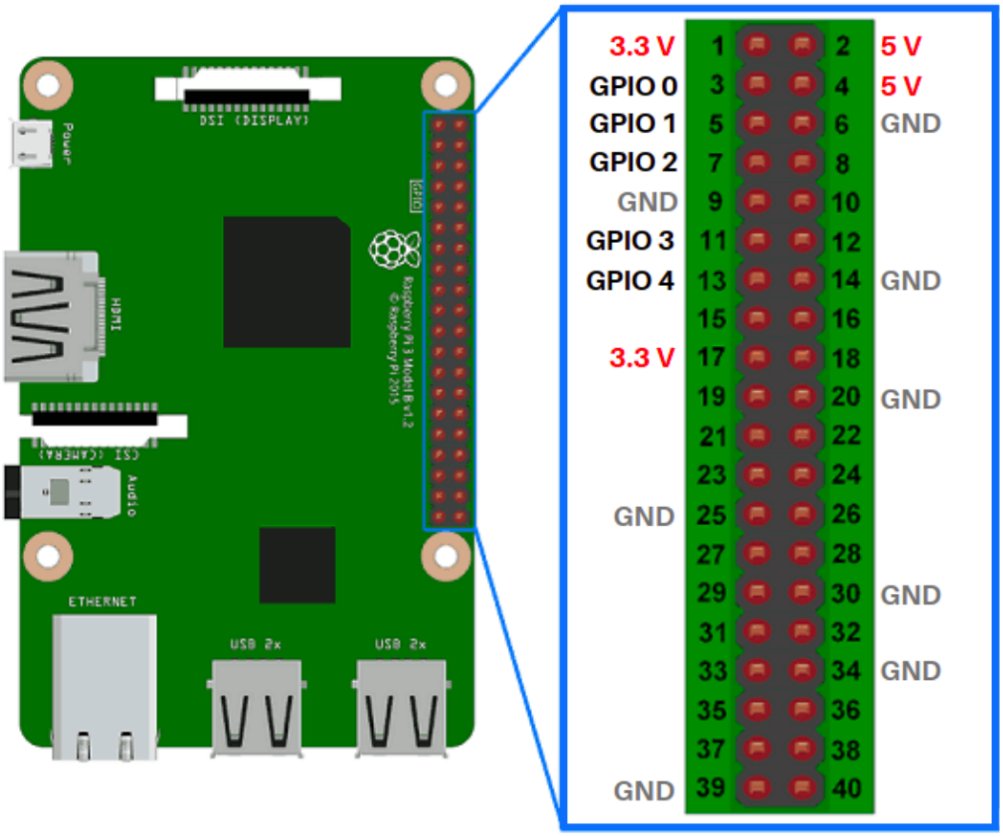
Am Anfang wurde eine C++ Implementation von CAN verwendet anstatt einer CANOpen Library, weshalb die Kommunikation nicht stattfinden konnte. Danach wurde auf das CANOpen Package in Python gewechselt. In der Dokumentation des Festo Roboters steht, dass, dass Bit 0x3020 verwendet werden muss. Doch bei der Python Implementierung des CANOpen Protokolls wird dies geändert auf Index 0 weshalb weder Daten gesendet noch empfangen werden konnten. Es wurde versucht auf eine Ethernet Verbindung umzusteigen, doch dies wurde nicht vom Roboter supportet.

5.2.2 Magnet

Anfangs haben wir nach starken Magneten gesucht, aber wir haben nur einen kleinen Kühlschrankmagnet gefunden. Hier war jedoch das Problem, dass wir ihn

nicht ausschalten könnten. Hier tauchte schon die erste Herausforderung auf. Wir mussten den Magneten an einem Servomotor montieren.

5.2.3 Raspberry Pi SSH



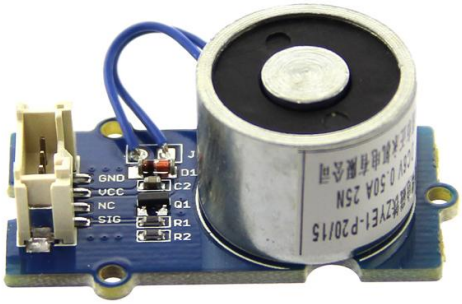
5.3 Zweiter Anlauf

5.3.1 Magnetverstärkung

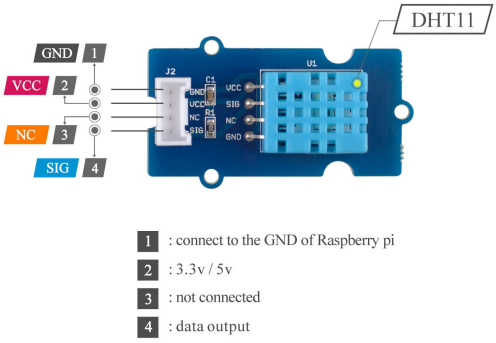
Da wir ohne einen GrovePi plus arbeiten, und direkt den Raspberry mit dem Elektromagneten verbinden, sind wir auf ein Problem gestoßen. Der Magnet lässt sich nicht anschalten. Unser Lösungsansatz war, die Stromversorgung separat durchzuführen und haben den VCC und GRND des Elektromagneten mit einem Spannungs- und Stromnetzteil versorgt und somit eine externe Stromfütterung erzielt.

Weiters kann der Elektromagnet keine Signale bekommen welche nicht auf 5 Volt sind, daher bräuchten wir ein Level Konverter oder eine andere Lösung.

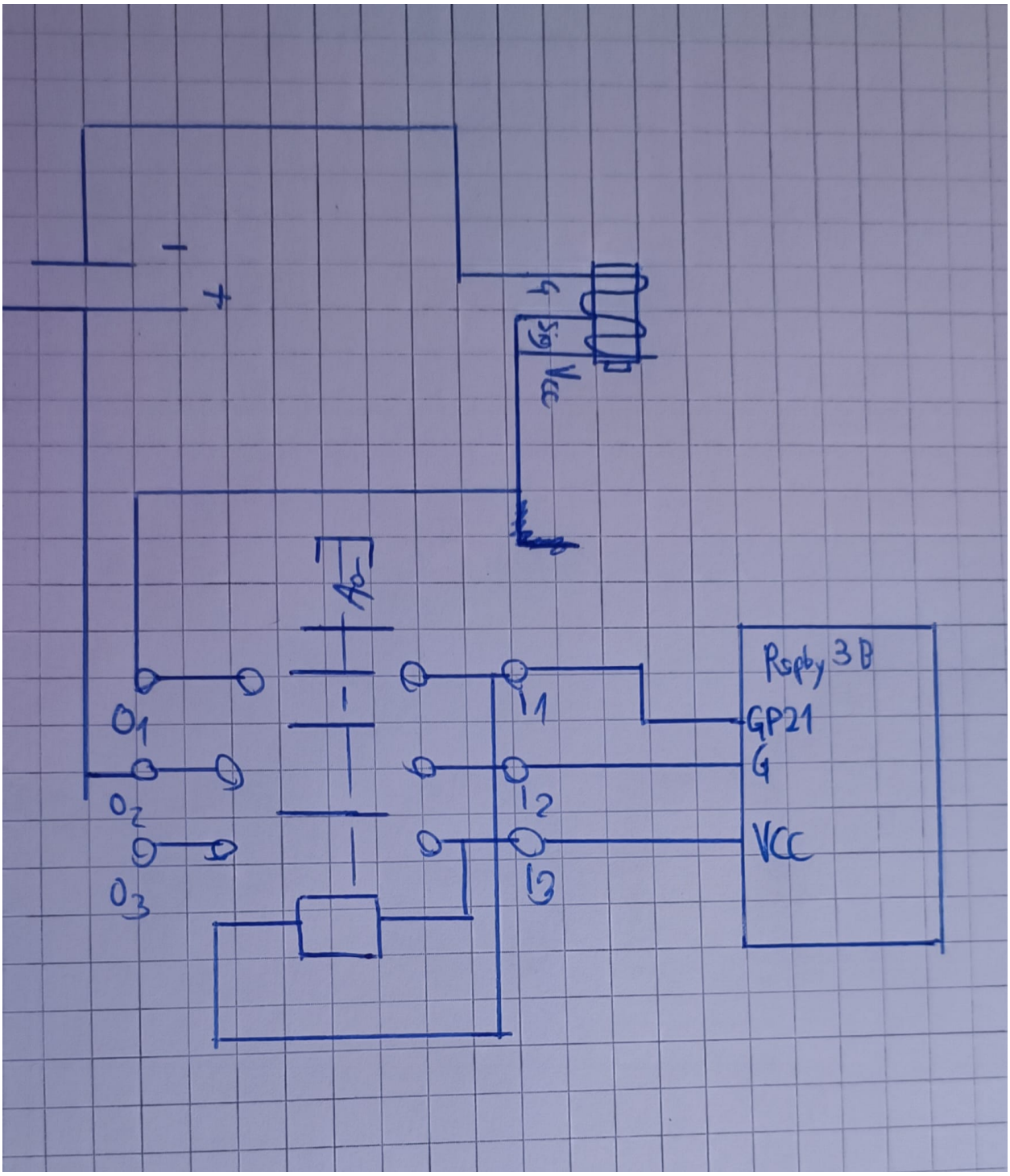
Unser Lösungsansatz → Mit Relay arbeiten, sodass wir den Magneten immer auf HIGH haben und sobald wir ihn Low setzen wollen, schalten wir ihn einfach aus.



Grove Elektromagnet



DHT11 Pin Out, was dem des Elektromagneten gleichkommt



Nachdem wir den Elektromagneten, so wie im Schaltplan auch zusehen ist, angeschlossen haben, haben wir versucht diesen anzusteuern.

Dabei haben wir folgendes Tutorial gefunden von der Herstellerseite.

Grove - Electromagnet | Seeed Studio Wiki
Grove - Electromagnet
https://wiki.seeedstudio.com/Grove-Electromagnet/

Dieses setzt allerdings voraus, dass man das zusätzliche GrovePi Plus Board auch hat. Da dieses nicht vorhanden war, haben wir uns entschieden einen alternativen Lösungsansatz zu probieren.

Dabei geben wir ihm einfach nur Strom oder nicht Strom. Das haben wir mit einem externen Netzteil und einem Relay geschafft.

```
             CAN
  SIGNAL     MODULE
   NAME  --- SIGNAL
    3,3V --- VCC
      5V --- Extra wire to Vcc TJA1050
     GND --- GND
     GND --- GND to CAN BUS (there is no GND terminal on CAN MODULE)
    MOSI --- SI
    MISO --- SO
    SCLK --- SCK/CLK
SPI0.CE0 --- CS
```

Neues Problem

Raspberry kann dem Relay keine 5V schicken, womit das Umschalten nicht funktioniert

Neue Lösung

Raspberry shield PiFace 2 verwenden, da wir somit ein Relay innerhalb des Raspberrys steuern können.

Nach weiterem versuchen konnte keine Verbindung über die CAN Schnittstelle hergestellt werden, weshalb auf die Digital I/O s gewechselt wurde. Der finale

CAN Code ist auf GitHub verfügbar.

Finaler Code

Der folgende Code ist die finale Version, welche zwar eine Verbindung herstellen konnte und ein Ready Signal schicken konnte, aber nicht reproduzierbar war.

Die letzte Version des Codes, um den Magneten zu aktivieren:

```
import time
import RPi.GPIO as GPIO

# Set up the GPIO mode
GPIO.setmode(GPIO.BCM)

# Define the GPIO pin to control the relay (for example, GPIO 1
electromagnet_pin = 21

# Set the pin as an output pin
GPIO.setup(electromagnet_pin, GPIO.OUT)

# Allow some time for setup
time.sleep(1)

while True:
    try:
        # Switch on the electromagnet by sending high signal to
        GPIO.output(electromagnet_pin, GPIO.HIGH)
        print("Electromagnet ON")
        print(f"GPIO State: {GPIO.input(electromagnet_pin)}")
        time.sleep(2)  # Keep the electromagnet on for 2 seconds

        # Switch off the electromagnet by sending low signal to
        GPIO.output(electromagnet_pin, GPIO.LOW)
```

```
        print("Electromagnet OFF")
        print(f"GPIO State: {GPIO.input(electromagnet_pin)}")
        time.sleep(5)  # Keep it off for 2 seconds

    except KeyboardInterrupt:
        # Clean up the GPIO pin state when the program is interr
        GPIO.output(electromagnet_pin, GPIO.LOW)  # Turn off the
        print("Program interrupted. Electromagnet turned OFF.")
        break
    except Exception as e:
        print(f"Error: {e}")
        time.sleep(1)  # Wait and try again

GPIO.cleanup()  # Reset GPIO pin states
```

Der finale Code, um mit dem Roboter zu kommunizieren und einen Zug zu machen:

```
import RPi.GPIO as GPIO
import time

#            R,  L,   U,  D
move_pins = [16, 16, 16, 16]
magnet_pin = 21
enable_pin = 26
ack_pin = 13
start_pin = 17

def setup_gpio():
    GPIO.setmode(GPIO.BCM)
    # Configure move pins
    for pin in move_pins:
        GPIO.setup(pin, GPIO.OUT)
        print(f"Setup pin {pin}" )
```

```
    # Configure input pins
    GPIO.setup(magnet_pin, GPIO.IN)
    GPIO.setup(enable_pin, GPIO.OUT)
    GPIO.setup(ack_pin, GPIO.IN)
    GPIO.setup(start_pin, GPIO.OUT)

def move(current_pos, start_pos, end_pos):
    make_move(current_pos, start_pos)
    # TODO: Magnet ON
    make_move(start_pos, end_pos)
    # TODO: Magnet OFF

def make_move(start, dest):
    # I. Go to the correct char lane (A, B, C, etc.)
    if start[0] != dest[0]:
        # 1. Go left or right?
        direction = 0 if ord(dest[0]) > ord(start[0]) else 1
        steps = abs(ord(dest[0]) - ord(start[0]))

        # Call method 'go' with direction (0 = right, 1 = left)
        go(direction, steps)

    # II. Go to the correct number lane (1, 2, 3, etc.)
    if start[1] != dest[1]:
        # 1. Go up or down?
        direction = 2 if ord(dest[1]) > ord(start[1]) else 3
        steps = abs(ord(dest[1]) - ord(start[1]))

        # Call method 'go' with direction (2 = up, 3 = down) and
        go(direction, steps)

def go(direction, steps):
    # Print the direction and steps for debugging purposes
    print(f"Direction: {direction}, Steps: {steps}")
```

```
    #for i in range(steps):
    GPIO.output(move_pins[direction], True)
    print("Move True")
    #time.sleep(5)
    GPIO.output(start_pin, True)
    print("Start true")
        #GPIO.output(move_pins[direction], False)
        #time.sleep(0.1)
        #GPIO.output(start_pin, False)

# Main execution
setup_gpio()

for pin in move_pins:
  GPIO.output(pin, False)

GPIO.output(enable_pin, False)
GPIO.output(start_pin, False)

current_pos = "A1"
start_pos = "A2"
end_pos = "B4"

GPIO.output(enable_pin, True)
time.sleep(20)
move(current_pos, start_pos, end_pos)

GPIO.output(enable_pin, False)
# Clean up GPIO pins when done
GPIO.cleanup()
```

6. Fazit/Lessons Learned

Unser Workflow begann mit der Aufteilung in zwei Teams, um paralleles Arbeiten und maximale Effizienz sicherzustellen. Gruppe A fokussierte sich auf die

Kommunikation zwischen dem Roboter und dem Raspberry Pi sowie auf das Mapping des Schachbretts. Gruppe B übernahm die Konstruktion des Schachbretts und die Implementierung des Elektromagnet-Mechanismus.

Zusammenfassung des Workflows:

1. **Schritt 1:** Erste Kommunikationsversuche über CAN-Protokoll in C++ und später Python, mit Problemen durch inkompatible Indizes und begrenzte Dokumentation. Folgend wurde der Magnet auf dem Servo Motor angebracht und der Code angepasst.
2. **Schritt 2:** Wechsel auf externe Stromversorgung und Steuerung des Elektromagneten mithilfe eines Relais, um Spannungsprobleme zu lösen.
3. **Schritt 3:** Integration des PiFace 2 für eine bessere Steuerung der Relais durch den Raspberry Pi.

Lessons Learned:

- **Dokumentation ist essenziell:** Unzureichende oder schwer verständliche Dokumentation führte zu Fehlinterpretationen und Problemen bei der Kommunikation mit dem Roboter. Zukünftig sollte vorab geprüft werden, ob verwendete Libraries mit der Hardware kompatibel sind.
- **Adaption is key**: Dieses Projekt hat uns gezeigt, dass in der Welt der industriellen Informationstechnologie nie alles auf einen Anhieb funktioniert und man oft auf alternative Ansätze wechseln muss. Daher haben wir gelernt, dass Anpassung sehr wichtig ist.
- **Hardware-Inkompatibilitäten beachten:** Der Einsatz zusätzlicher Hardware (z. B. GrovePi Plus) sollte frühzeitig geplant und evaluiert werden, um Verzögerungen durch Workarounds zu vermeiden.
- **Den Motor zum Laufen bringen, bevor man das Auto lackiert:** Wir haben gemerkt, dass es ein guter Lösungsansatz ist, zuerst einmal etwas zum Laufen zu bringen, bevor man sich auf die Technologie fixiert und den Rest implementiert.

Unser Fazit zeigt, dass sorgfältige Planung, Flexibilität und eine iterative Herangehensweise entscheidend für den Erfolg technischer Projekte sind. Die dokumentierten Herausforderungen und Lösungen können zukünftigen Projekten als Leitfaden dienen, um ähnliche Probleme zu vermeiden.