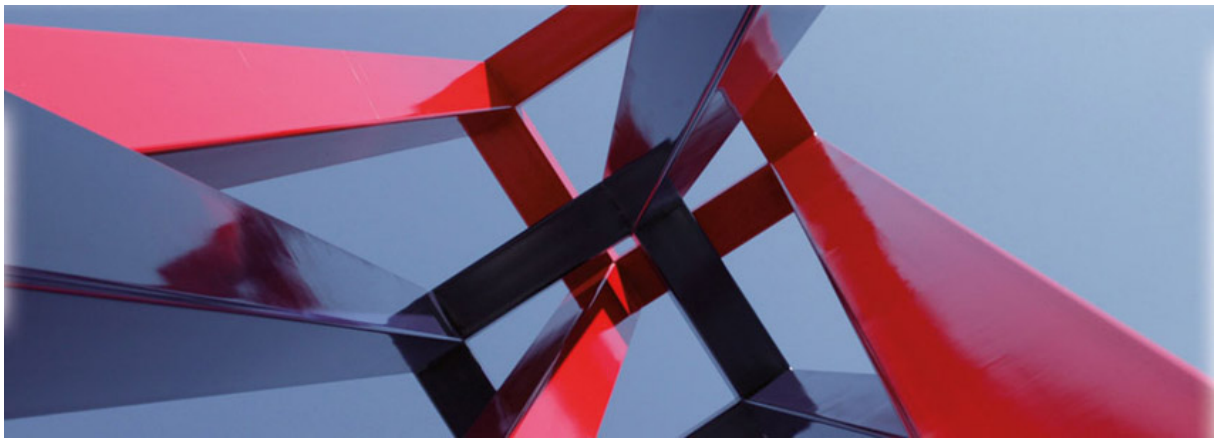
Hochschule Hof
University of Applied Sciences

Artificial Intelligence in Robotics - WS2022



As we progress further into the 21st century, the importance of artificial intelligence (AI) and robotics becomes increasingly evident. These technologies have the potential to revolutionize industries, improve the quality of life, and drive human progress to unprecedented levels. With their growing influence, it is crucial to understand the significance of AI and robotics in shaping our future.
For economic growth and job creation, AI and robotics are transforming industries by automating repetitive tasks, increasing productivity, and reducing operational costs. While there are concerns about job displacement, these technologies have the potential to create new job opportunities in emerging sectors, such as AI and robotics engineering, data science, and human-machine interaction design.

In healthcare, AI and robotics are revolutionizing healthcare by enabling early diagnosis, personalized treatment, and improved patient care. From medical image analysis and drug discovery to robotic surgery and rehabilitation, these technologies are enhancing the efficiency and effectiveness of healthcare services while reducing the burden on healthcare professionals.
These are only a few examples. Therefore, the need for experts is constantly growing. In the course ‘ai in robotics’ students implemented various projects, where they used state of the art methods from ai and robotics.

Christian Groth (publisher)
University of applied sciences Hof
2023
DOI 10.57944/1051-136

Table of contents

Creating a Chess-Playing Robot Using Python and a Nyrio Ned 2

1th Sebastian Rainer Bär
Department of Robotics
Hof University of Applied Sciences
Hof, Germany
sebastian.baer.2@hof-university.de

2th Johannes Matus
Department of Robotics
Hof University of Applied Sciences
Hof, Germany
johannes.matus@hof-university.de

3th David Samuel Weiß
Department of Robotics
Hof University of Applied Sciences
Hof, Germany
david.weiss@hof-university.de

4th Dejan Reinhardt Fraas
Department of Robotics
Hof University of Applied Sciences
Hof, Germany
dejan.fraas@hof-university.de

5th Eugen Kudraschow
Department of Robotics
Hof University of Applied Sciences
Hof, Germany
eugen.kudraschow@hof-university.de

6th Ashanga Wayi Elvis
Department of Robotics
Hof University of Applied Sciences
Hof, Germany
ashanga.elvis@hof-university.de

***Abstract*—Implementation of a chess playing robot with visual representation of the board, the ability of the robot to make smart decisions regarding the current game status based on AI, the option to input human moves by speech and to train the algorithm for different chess openings.**

***Index Terms*—nyrio ned 2, chess, chess engine, chess algorithm, speech recognition, object grabbing, python, stockfish chess engine, stockfish**

I. Introduction

This paper presents a method for implementing a chess game in Python for a Nyrio NED 2[7] robot to play against a human opponent on a physical board. The human player may input their moves through a graphical user interface or by speaking, and the computer must then determine the optimal response based on the current state of the digital board. To achieve this, the project may utilize an in-house chess AI or the open-source Stockfish[4] engine to calculate the robot's next move. Once this calculation is complete, the robot must be provided with the necessary information to pick up and move the appropriate chess pieces, including details on the type of move (e.g. normal, capturing, en passant, castling) and the types of the pieces being moved.

II. APPROACH / CONTRIBUTION

A. Hypothesis

The hypothesis of this paper is that it is feasible to design and implement a chess game in Python for a Nyrio NED 2 [7] robot to play against a human opponent on a physical board. The computer system must be able to accurately analyze the current state of the digital board and determine the optimal response to the human player's moves, and the robot must be able to execute the chosen move by accurately manipulating the necessary chess pieces.

B. Methodology

1) Game Interaction: To represent the current state of the chess board visually, pygame[1] is used to draw the current board to the screen and make it possible to interact with the board. Further information about the game, like the turn of the player or made moves and AI decisions are output to the console. The graphical user interface can be utilized to move pieces by clicking on a piece on a square and afterwards clicking on the square the piece should be moved to.
During the Move of the robot the board is not responsive and does not accept input. The board is drawn based on a 2D Array representation, which is produced by a Gamestate class. In addition the human player can reset the game, undo moves, record own opening strategies or input moves via speech through pressing keys on the keyboard. The move of the robot also needs to be triggered manually by pressing the corresponding key after the human player input their move. The already mentioned Gamestate class is used to wrap the python chess library[2] to handle game logic. The Gamestate class provides methods for making moves, interacting with the chess-AI, and telling the robot which moves to execute. It processes the user input and mediates between robot AI and the user. It also provides functions for converting between chess notation and tuple coordinates, and for accepting natural language descriptions of moves. The GameState class itself provides the representation of the chess board in a format the GUI can interpret, based on the chess library.
To utilize the open source chess engine Stockfish[4], the Script additionally uses the python stockfish wrapper library[3]. The Stockfish engine[4] itself must be installed separately on the system and the Path additionally needs to be specified within the code. Alternatively the self made chess-AI can be utilized.

2) AI/Chess Engine: The chess AI is an implementation to calculate the best possible next chess move. It defines some

constants, including values that assign a numeric value to each chess piece. The AI operates on the current state of a chess game.

In order to improve the performance of the chess AI during the opening phase of the game, we have implemented a system that allows the user to record and save specific opening strategies with a maximum of 20 moves. This is achieved through the use of a database, specifically a CSV file, which is accessed and modified through a Database Connector. The user can initiate the recording of an opening strategy by pressing a designated key, and then proceed to make the desired opening moves.

These moves are then stored in the database for future reference. While the use of a CSV file for storing the recorded opening strategies may not be the most efficient method in the case of a large database, the current size of the database does not significantly impact performance. To counter this issue, a maximum of 20 moves per opening was implemented in the opening database. As a result, after 20 moves, the opening database does not need to be checked and the AI can make its move more efficiently. Additionally, the use of a CSV file allows for easy modification of the database system in the future if needed, simply by modifying the Database Connector. Basic and important opening strategies have already been included in the database. If no opening move can be found, it uses the negamax algorithm with alpha beta pruning to determine the best move.

The negamax algorithm is a search algorithm used in two-player games to find the best move for a player. It works by assuming that the opponent will always choose the move that is best for them, and therefore the player should choose the move that is worst for the opponent.

The negamax algorithm is a recursive function that calculates the best score that can be achieved from that state. It searches through all possible moves and their resulting game states, and assigns a score to each one based on how good it is for the player. It then returns the maximum of these scores, since the player will always choose the move that is best for them.

The current maximum possible depth is 3, meaning the AI looks 3 moves ahead. Values above 3 would result in performance issues and higher waiting times when calculating the best next move.

Alpha-beta pruning is a technique used to improve the efficiency by cutting off branches of the search tree that cannot possibly produce a better score than what has already been found. It works by maintaining two values, alpha and beta, which represent the best score that the current player and the opponent can achieve, respectively. If the score of a move is greater than or equal to beta, the search can be stopped, since the opponent will not choose a move that is worse for them than what they already have. Similarly, if the score of a move is less than or equal to alpha, the search can be stopped. This allows the search to avoid exploring unnecessary branches.

The board is scored based on the current position of the chess pieces. If the position is a checkmate, it returns a high constant to indicate the significant positional advantage or disadvantage of a move. Otherwise, it returns the material score of the position, which is calculated by summing the values of all the pieces on the board using the values of the chess pieces (pawn: 1 point, knight: 3 points, bishop: 3 points, rook: 5 points, queen: 9 points, king: infinite points).

It also includes positional bonuses for each piece, which are taken from arrays where the positions for every type of piece are scored (Pawns and knights are more valuable in the middle of the board for example). The positional bonuses are multiplied by a constant value to tweak the weight of the positions. By adjusting the value, it is controllable how much of an effect the positional bonuses have on the final score. If the value is set to a high value, the positional bonuses will have a greater impact on the final score, while if it is set to a low value, the positional bonuses will have a smaller impact.

3) Speech Recognition: Several speech recognition libraries were tested. Finally, we chose the Python library Speech Recognition [5] to process audio input, since some tried speech recognition tools are either not available for free or did not fit our needs. PyAudio[6] is used to record audio input.

Since speech recognition does not always accurately generate what is spoken, the spoken audio signal must first be filtered to avoid confusion of the spoken expression. The corresponding format must be a string with the start and end coordinates of the move. Furthermore, speech recognition also supports the game mechanics castling and pawn promotion. When recording a move, a sound is played to indicate that the recording is starting. After the recording is done, another sound is played to show that the recording is finished. If the recording is not recognized an error sound is played instead.

Speech recognition starts when it is the player's turn. In addition, the "a" key must be pressed. After the key is pressed, a sound is played, which signals that recording for speech recognition is starting. Voice command must be given within 1.5 seconds, otherwise the recording will be stopped. During recording, the player has 5 seconds to define the voice command. After recording, the resulting string is then analyzed and filtered for keywords. If these keywords are not present or the voice command is not correctly understood, nothing is passed to the game logic and a sound is played which signals that the move is invalid.

The move string for a valid move can include the keywords:

- For a valid move: "from" and "to"
- For casting: "side"
- For the pawn promotion the respective figure to promote to

Due to the different pronunciation of words and numbers, incorrect interpretations can occur. Thus leading to an invalid move. To minimize these errors, words that sound similar, such as "form" and "from", and incorrectly interpreted numbers are stored in a library so that they can be filtered afterwards. Speech recognizer matches the confusing words to the correct key and replaces it. Finally, a string in the chess notation

format, e.g. ”a2a3”, is created.
The accuracy of the given input heavily depends on different circumstances such as audio quality, environmental noises, the duration of the record, the pronunciation of the individual along with the context it is transmitted to. In order to avoid any possibility of confusion the record has to be evaluated and possibly replaced by the right keyword that is necessary for further processing.
To ensure that the game logic accepts the edited format, several moves were tested. Any mispronunciations that were leading to confusion were added to the library to make speech recognition more reliable. A quality microphone also mitigates confusion of the spoken words.

4) Robot: The Robot we are working with is the Niryo Ned 2 [7]. To control the robot we have used the Python library Pyniryo [8].

Initially the robot should be controlled via WiFi. Due to the fact that the speech recognition needs an internet connection and two wireless connections are not possible, we had to make a physical connection with the robot via Ethernet.

In general the robot is capable of playing either color, which can be determined at the start of the game. The robot object gets initialized depending on the choice, true for playing with white and false for playing with black.

Because we do not have any visual object detection we have resorted to the realities of the chessboard. The length of the board are for the outside 38cm and for the field 32.5 cm. The length of one square is 4cm. Our pieces heights are in the range from four to eight centimeters. Therefore we have determined a fixed entry point for the robot centralized at the side of the chessboard, depending on the color the robot is playing. From here on the piece coordinates do not differ and could be found out by a trial and error approach. The fields for the captured pieces and the substitute queen for promotion are next to the chessboard, depending on the color the robot is playing with. We have used an array with 15 different spots where for each capture the index of the array gets incremented by one. This was managed by implementing a Chess-Fields class which contains the necessary information.

For the moves we had to cover the following possibilities: Moving a piece without a capture, capture a piece by moving it on the side of the board and moving the capturing piece to the destination square afterwards, castle rook and king depending on the side (Queenside, Kingside Castling), en passant captures.

To facilitate the proper tracking of promoted pawns during a chess game, we have implemented a system that converts their moves to those of a pawn for the purposes of the robot determining the right physical piece to grab, while maintaining the FEN (Forsyth-Edwards Notation) representation of the promoted piece for the AI. This is necessary because, in the context of the program, it is not possible to physically substitute the promoted pawns with the piece they were promoted to, but the robot needs to grab the right physical piece even if it was promoted to another one.

In order to accurately track the positions and movements of promoted pawns, an array is used to store and update positions of promoted pieces while they get handled like the pieces they were promoted to by the Gamestate. This allows the program to maintain an accurate representation of the positions and movements of promoted pawns while preserving the appropriate FEN representation.

To successfully grab and place pieces without them getting out of alignment, we used a chess-set where the pieces had sharp edges. This gave the opportunity to pick every piece either from the x-angle or the y-angle.
If a piece is picked up from the board the robot needs to know the height of the piece so that the gripper has the right angle.
If a piece is placed on the board, or beside it due to capture, the robot needs to know the height of the piece so that a collision of the picked up piece and the target ground is avoided. This was managed by implementing a class which contains all the necessary information in order to do so. Due to the fact that the height of the pieces differ, especially the queen and the king, the gripper could not reach the minor piece without collision. We had to implement an algorithm that turns the gripper either when picking or placing a piece if it stands next to a king or a queen on the x-axis. The code for the robot checks whether the king or the queen is standing or will stand next to the target field. If so the gripper is turned by 90 degrees and turns back to default after success.

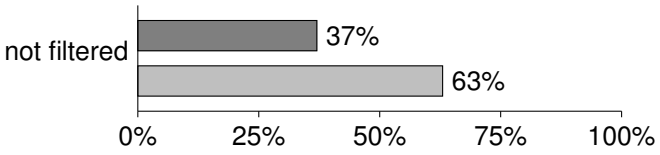
III. Experiments

A. Description

The experiment aims to evaluate the effectiveness of filtering specific words in minimizing the error rate of speech commands. The experiment is divided into two parts. First, various possible moves are spoken multiple times and processed without filtering. In the second experimental setup, a particularly error-prone move is spoken and processed with filtering, followed by the same move spoken again without filtering. The results are then compared to assess the extent to which filtering improves the output. Filtering is necessary as there is often misinterpretation of words by the Google API. Prior to the start of the experiment, certain restrictions were established to ensure better test results, such as ensuring that the same speaker speaks the moves under identical noise conditions. The Python library ”Speech Recognition” was used to perform the experiments. A custom-written library was used for filtering, which replaces misinterpreted words. The commands were spoken through a headset.

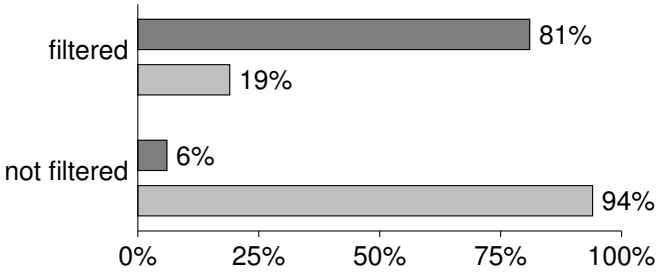
B. Experiment I

For our test data, we first attempted various moves without any filtering. The moves were spoken multiple times. Our dataset was limited to the following moves: [A2A3, B2B3, C2C3, D2D3, E2E3, F2F3, G2G3, H2H3, A2A4, B2B4, C2C4, D2D4, E2E4, F2F4, G2G4, H2H4]. During the implementation, 63% of the test cases failed.



C. Experiment II

The move H2H3 was particularly prone to errors. We conducted some tests both with and without filtering to determine whether filtering could improve the accuracy of the determined move. The filtering resulted in a correct outcome in 81% of the test cases. In contrast, only 6% of the test cases without filtering yielded a correct outcome.



D. Conclusion

The experiment aimed to evaluate the effectiveness of filtering specific words in minimizing the error rate of speech commands. The results showed that filtering significantly improves speech recognition accuracy, particularly for error-prone moves like H2H3 and H2H4. The use of filtering resulted in a 81% correct outcomes compared to 6% without filtering. The experiment was conducted under controlled conditions using Python. In conclusion, filtering improves speech recognition and results in a smoother and more efficient game flow.

IV. CONCLUSION / FUTURE WORK

In conclusion, the discussed project aimed to develop a chess-playing robot that could play against a human opponent on a physical board. Through the implementation of a chess AI, the robot was able to analyze the current state of the board and determine the optimal response to the human player's moves. The robot was then able to execute the chosen move by manipulating the appropriate chess pieces. The project was successful in demonstrating the feasibility of creating a chess-playing robot using a combination of software and hardware, and in evaluating the performance and capabilities of the resulting system. The project has shown that it is possible to build a functional and effective chess-playing robot that can provide an engaging and challenging experience for human opponents.

During the implementation of the project, we realized that the quality and performance of the implemented chess AI depends heavily on the method that evaluates the board after each move. To make this more precise, an efficient way would have to be found to include possible attacks or defenses of the pieces and to evaluate the board in more detail. So far, this has caused considerable performance losses and was therefore excluded from the final program. Additionally a different database system could potentially be implemented to improve search performance for chess openings. Using Stockfish does eliminate those problems.

Furthermore the package used for speech recognition requires internet connection. It is possible to use the API offline. However some additional files are needed that take up too much memory. Making speech recognition available offline without heavily increasing the project size would be a beneficial addition.
In addition to that, implementing a camera system to track the positions of the pieces on the board would be a useful addition to the project. This could potentially improve the efficiency of the system by eliminating the need for manual input of moves and allowing the computer to directly observe the state of the physical board. Overall, incorporating a camera system into the project could bring several benefits and enhance the capabilities of the system.

V. REFERENCES

1 https://www.pygame.org/news
2 https://python-chess.readthedocs.io/en/latest/
3 https://pypi.org/project/stockfish/
4 https://stockfishchess.org/
5 https://pypi.org/project/SpeechRecognition/
6 https://pypi.org/project/PyAudio/
7 https://niryo.com/product/ned-2-education-robotics-arm/
8 https://docs.niryo.com/dev/pyniryo/v1.1.2/en/index.html

Herbie, the self driving RC Car

1st Sebastian Peschke
MC 5
Hof University
Hof, Germany
speschke@hof-university.de

2nd Ron Polenthon
MC 5
Hof University
Hof, Germany
rpolenthon@hof-university.de

3rd Hannes Steinel
MC 5
Hof University
Hof, Germany
hsteinel@hof-university.de

4th Johannes Strecker
MC 5
Hof University
Münchberg, Germany
jstrecker@hof-university.de

***Abstract*—This document is an introduction to our self-driving RC car Herbie. This vehicle works with a 360° microphone, HD camera and an Nvidia Jetson as processor. This vehicle waits for sound in its range and follows a detected person. The detection works with an objection detection AI model. In this paper we explain how we built and programmed the car.**

***Index Terms*—robotics, computer vision, speech recognition, self-driving, python, jetson**

I. State of the Art

In this paper we will explain how we built Herbie, our self-driving RC car. In order for this to work we had to first assemble the car, and then we had to also implement the self-driving mechanism.

Most of the hardware we used is readily available and just had to be connected together. The base of our car is a Red Cat Blackout RC car. We used the provided AI-Car STL files to 3D print all the parts we needed to mount our components to Herbie.

As the main logic board for the project we used a Jetson Xavier NX Developer Kit. To control the cars' motors, we utilized an Adafruit PCA9685 16-channel servo driver.

The microphone we worked with is a ReSpeaker Mic Array v2.0. We used two different cameras throughout the project. The first is an Intel Realsense Depth Camera D435 and the second a generic Logitech webcam.

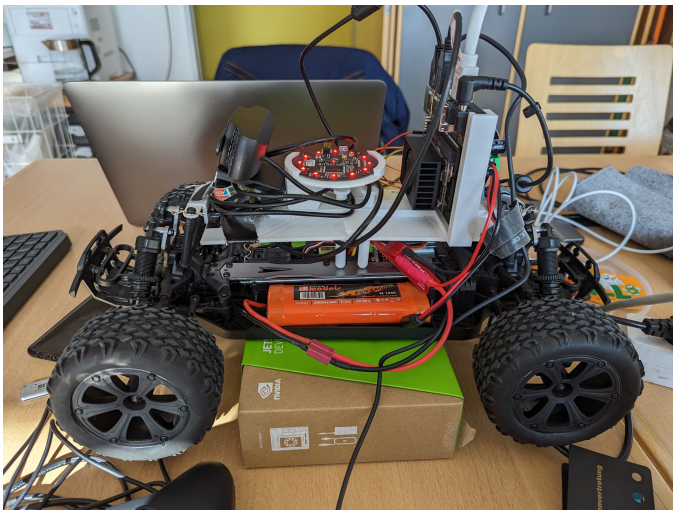


Fig. 1. The completed car

For the software we tried to use mostly pre-made code snippets but ended up having to do a lot of coding on our own.

Since we used an NVIDIA Jetson as our compute unit, we used L4T [1], a proprietary Ubuntu based OS from NVIDIA, as our operating system. Our programming language of choice for this project was Python because there are a lot of useful libraries for the parts we have.

We utilized the ReSpeaker's library [2] for example to automatically locate from which direction a speakers voice comes from and also to control the RGB LED ring around the microphone [3].

To control the car we used both the Adafruit PCA9685 [4] and servo kit [5] libraries.

We also utilized a pre-trained roboflow model [6] for our object detection and the roboflow inference server docker container to run the model locally on the Jetson.

In this paper we will not only show how we built and programmed Herbie, but also how to use and recreate it.

II. Approach

A. Goal

Our goal for this semester was the creation of a scale model of a self-driving car. The car should be able to listen to a wake-up word for activation and automatically drive towards whoever is speaking, whilst avoiding obstacles.

During the project however, our goals have changed a little because of a shift in focus to certain aspects, such as object detection.

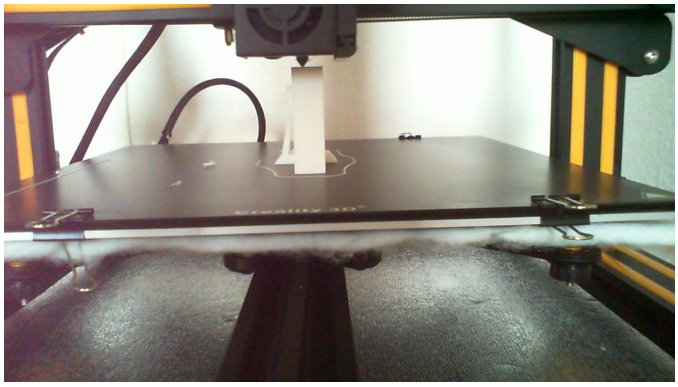


Fig. 2. 3D printing parts for Herbie

B. The hardware

As a base for our self-driving car we used a default RC-car. To set everything in place we 3D-printed ("Fig. 2") mounts for our camera, microphone and the processor.

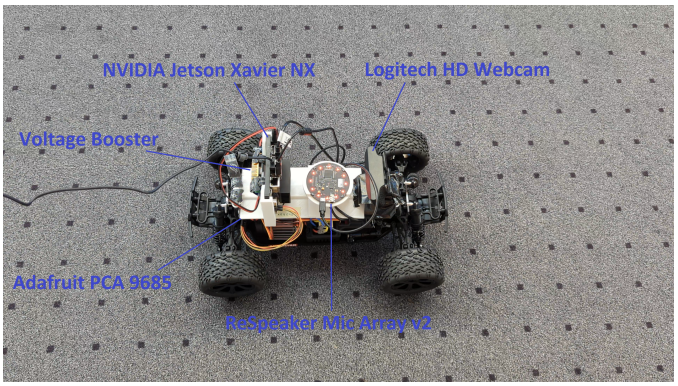


Fig. 3. The complete hardware of Herbie

An Nvidia Jetson Xavier NX Developer Kit is used as the main processor. The Jetson Xavier NX Developer Kit is a powerful tool for developers.

It offers a comprehensive set of features, including an 8-core ARM CPU, a 512-core Volta GPU, and a wide range of I/O options. With this kit, developers can create applications that take advantage of the latest AI and deep learning technologies.

Additionally, it provides a robust platform for developing and deploying embedded systems.

The camera is a default Logitech HD Webcam. In our case this was the right choice because we don't need high quality output or special features like depth detection.

We used a ReSpeaker Mic Array v2.0 as the microphone. It features four microphones arranged in a circular array, providing 360° sound capture capability and noise cancellation.

Additionally, it has an integrated LED ring that can be used for visual feedback. We also used an XBOX One Controller as a remote control for the car and to switch between different operating modes.

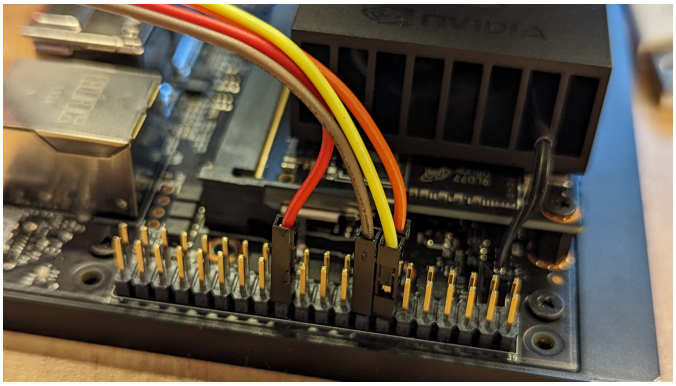


Fig. 4. The daughter-board connection on the Jetson

After a couple of experiments we figured out how to connect the Jetson to our daughter-board in order to control the motors.

As you can see in "Fig. 4" we connected the i^{2}c connectors as follows:

- Ground(brown) to Pin 25
- SCL(orange) to Pin 28
- SDA(yellow) to Pin 27
- VCC(red) to Pin 17

We used an Adafruit PCA9685 16-channel servo driver as an i^{2}c daughter-board to control both the main drive motor and the servo motor for steering.

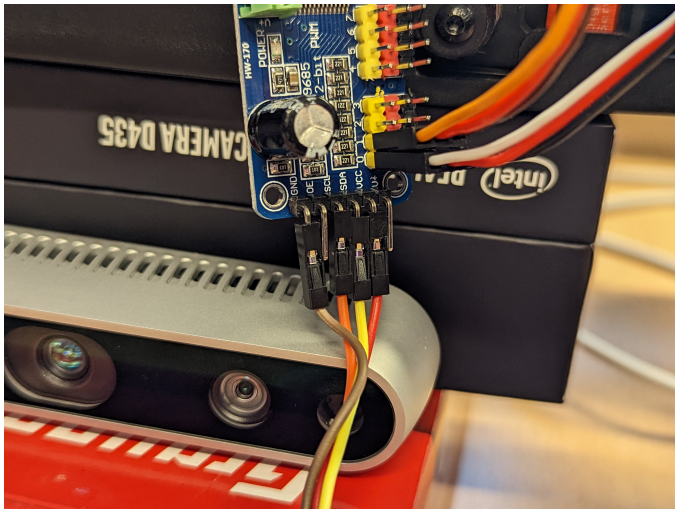


Fig. 5. The motor connections to the daughter-board

As shown in "Fig. 5", we connected the servo motor to channel 1 on the PCA9685 and the drive motor to channel 0.

C. The software

The biggest problem we had to solve was the software for Herbie, since we didn't only have to program the RC functionality from scratch, but we also had to add voice control and object detection to the car. This part of the project took by far the most time.

In the following sections we will go into detail how we solved each of the three different main program parts: driving, speech recognition and object detection. At the end we will also explain how those program parts come together and interact with each other.

1) Driving: Since we connected our motors to an Adafruit PCA9685 we were able to use the Adafruit ServoKit for Python to control the car. Whenever the drive of our car gets initialized, we first set up the main drive motor as a continuous servo and the steering servo as a standard servo motor.

This way we can use the built-in ServoKit functions `throttle` and `angle` with the motor and servo respectively. To further simplify the control structure we defined two methods `setSpeed()` and `setDirection()`.

The first normalizes the speed control from -1 (full speed reverse) to 1 (full speed forward) with 0 being full stop. The second method normalizes steering from -1 (full left) to 1 (full

right) and translates those directions to angles for the servo motor.

2) Speech Recognition: After experimenting with multiple options for speech recognition as described in "Experiments"(Chap. III-C), we settled on using the built-in function of the ReSpeaker microphone array to calculate, where the speech is coming from.

We used this data to divide the 360° around the car into 4 sectors:

- "in front": don't move
- "to the left": turn 90° left
- "to the right": turn 90° right
- "behind": turn 180°

In order to correctly respond to those cases, we programmed three different methods, which turn the car by 90°'s left, right or 180° by turning and then driving for the exactly correct amount of time. After Herbie detects voice and automatically turns into the direction of the sound, it switches itself into the "Object Detection Mode".

3) Object Detection: After a lot of trial and error we found the roboflow inference server docker container for our object detection. This way we could easily run any roboflow model directly on the Jetson's GPU and quickly query for results via the roboflow API running in the docker container.

The API also returns any detections as a simple JSON object, which we can use in our Python codebase. We currently use the pre-trained roboflow model "Human v2"[6] to recognize people in front of the car and follow them around.

To compute the output json from the model we used an easy algorithm. The json automatically detects the center of the detected object.

Fig. 6. Example image for the object detection - 1

Fig. 7. Example image for the object detection - 2

We calculate the difference of the y coordinate from the center of the image to the center of the object. If this value has an offset more than 10% we set the speed of the car directly to the percentage of the y center value.

The reason for that is that if a person is further away the camera sees more space between the object and the top of the image. If the object fills the whole image in the y-axis, the person is right in front of the car, and it can set the speed to zero.

The x value is in range between 0 (left side) and 1 (right side). The value is the input for this formula:

$$tanh((x * 2.0) - 1.0) * 0.25$$

This formula creates the following graph.

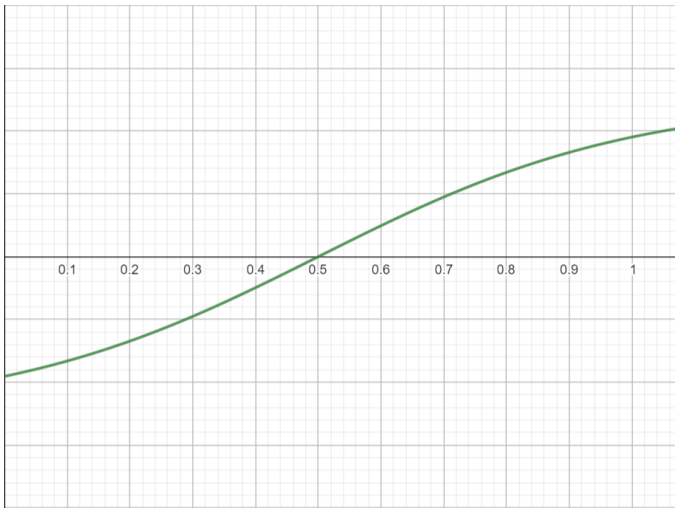


Fig. 8. The graph for the speed calculation

4) Bringing it all together: In order to bring all the different pieces of software together, we had to write a main Python script. Here we defined four different "modes" for Herbie:

- Mode “stop”, Herbie cannot move and can be restarted or turned off.
- Mode “controller”, Herbie can be controlled by an attached XBOX One controller.
- Mode “voice”, Herbie listens to nearby voices and tries to follow them.
- Mode “camera”, Herbie uses object recognition to follow humans.

After Herbie starts, it is in mode “stop”. In order for us to switch between the different modes, we utilized the ABXY buttons on the connected Controller.

The Y-button switches into mode “controller” and turns the LEDs yellow. The X-button turns on the “voice” mode and the LEDs blue. The A-button switches Herbie into “camera” mode and turns the LEDs green. The B-button is a sort-of “emergency-off” and switches Herbie into the “stop” mode which stops the car.

In this mode we can restart our `herbieStartup` system service with the start button and shut down the system completely with the back button.

III. Experiments

Quite early on in our project we realized that different Python version requirements for the microphone, camera and drive controller meant that we could not rely on a lot of premade code.

To understand each of the problems we faced, our group decided to run a couple of experiments to figure out which part would work with which Python version and library.

A. The correct L4T version

The first experiment we ran was to try different Linux for Tegra versions. In the beginning of our project we tried to run the newest version L4T 35.1.

Unfortunately we soon ran into problems, since the Intel realsense library was only supported up to L4T version 32.4.3. This meant that we had to switch our NVIDIA Xavier NX development board to L4T version 32.4.3 and Jetpack version 4.4.

B. Dead Hardware

Unfortunately we could not complete the project without a bit of dead hardware. One experiment we did partake in involuntarily happened, when the Jetson got unplugged whilst we were installing some software.

This resulted in a corrupted installation of Linux as well as a damaged SD-card and SSD. Fortunately we managed to reinstall all the lost software and configuration despite not having a backup. We accomplished this by mounting the corrupted SD-card on a different PC and copying the bash history

C. Speech Recognition

1) PicoVoice: The initial idea for our speech recognition was to have an AI detect a “wake-up-word”. After a lot of research we first tried to work with PicoVoice.

This approach would have worked as a local service on the Jetson. The implementation is very efficient, and we would have been able to process the audio feed directly without any indirection.

However, after some experimenting we found out that PicoVoice only works on Jetson Nano and doesn’t support the CPU architecture of the Xavier NX.

2) OpenAi Whisper: Soon after we tried OpenAI Whisper, which is only able to process audio files and no ongoing microphone streams.

So in order to work with this particular technology we had to write a script that converts a given audio stream into separate audio files. After a pause or after silence an audio file is generated so that coherent sentences or commands can be issued.

This approach of creating individual audio files that can be evaluated afterwards worked pretty well. Unfortunately the transcribing of audio files with the Whisper Tiny Model (the smallest possible model for more performance) took about five to twelve seconds.

With that in hand we decided to take another approach and keep this option as a backup plan if all else fails.

3) AssemblyAI: The last experiment was an attempt with AssemblyAI. We tried to outsource the payload of detection and computation with a cloud based solution.

The problem with AssemblyAI was that there were some version conflicts with our Ubuntu and Python versions. Because of that we were not able to use the local library of AssemblyAI.

IV. Conclusion and Future work

After a long project we can conclude that despite multiple problems we still managed to partly fulfill our goal to build a working, self-driving car. Our work of gathering information to assemble and build the car was a huge success.

We were able to get our hands on various technologies including the base board NVIDIA Jetson, the ReSpeaker Mic Array v2.0, object detection with a RealSense camera and the 3D-printer to mount it all together on the RC-car.

We used the RC-car as a base and got everything to communicate with each other.

The proprietary NVIDIA Ubuntu image as operating system helped us as an interface to communicate to the single parts as well as process the data we got from the peripherals and develop an intelligent car that reacts to its surroundings.

The experiments have shown that versioning and finding fitting technologies to suit our requirements is a big issue in today’s robotics. Various technologies like PicoVoice also show that availability for other platforms have to be engineered carefully and with enough foresight to not be trapped in a state of debugging and research which can be very time-consuming.

A further conclusion that is important to be drawn is that one should always operate with care and caution to not break components or even the entire operating system when working with hardware on low-level.

The only thing left to do is to really let the car drive on its own. The final implementation and further experiments and adjustments concerning that matter will be referenced in our future work and papers.

References

[1] NVIDIA. "Jetson linux." (Feb. 2, 2023), [Online]. Available: https://developer.nvidia.com/embedded/jetson-linux (visited on 02/12/2023).

[2] J. Shao, jerryyip, and fanjm95. "Respeakerd." (Apr. 8, 2019), [Online]. Available: https://github.com/respeaker/respeakerd (visited on 02/12/2023).

[3] Y. Xiong, B. Zuo, J. Shao, HinTak, and jerryyip. "Pixel ring." (Jun. 24, 2021), [Online]. Available: https://github.com/respeaker/pixel_ring (visited on 02/12/2023).

[4] E. Herrada, sommersoft, A. Delaney, *et al.* "Adafruit pca9685." (Jan. 20, 2023), [Online]. Available: https://github.com/adafruit/Adafruit_CircuitPython_PCA9685 (visited on 02/12/2023).

[5] E. Herrada, A. Delaney, Kattni, *et al.* "Adafruit servokit." (Jan. 20, 2023), [Online]. Available: https://github.com/adafruit/Adafruit_CircuitPython_ServoKit (visited on 02/12/2023).

[6] H. v2, *Human dataset v2 dataset*, https://universe.roboflow.com/human-v2/human-dataset-v2, Open Source Dataset, visited on 2023-02-12, Apr. 2022. [Online]. Available: https://universe.roboflow.com/human-v2/human-dataset-v2.

Creation process of a protoype for autonomous driving in the Formula Student

Jannik Maisch
Fakultät Informatik
Hochschule für angewandte Wissenschaften Hof
Hof, Germany
jannik.maisch@hof-university.de

Max Kruggel
Fakultät Informatik
Hochschule für angewandte Wissenschaften Hof
Hof, Germany
max.kruggel@hof-university.de

Christoph Kreutzer
Fakultät Informatik
Hochschule für angewandte Wissenschaften Hof
Hof, Germany
christoph.kreutzer@hof-university.de

Tim Ordnung
Fakultät Informatik
Hochschule für angewandte Wissenschaften Hof
Hof, Germany
tim.ordnung@hof-university.de

Benjamin Gruber
Fakultät Informatik
Hochschule für angewandte Wissenschaften Hof
Hof, Germany
benjamin.gruber@hof-university.de

***Abstract*—Automatization does not lie in the future. In fact, it is already implemented in many kinds of processes in various parts of the industry. E.g., in the assembly of car parts or evaluation of test results. Autonomous driving, a form of automatization, is a field which is still in its infancy. Some car companies like BMW, Mercedes and Tesla achieved some noteworthy improvements in the last couple of years, but it is still far from perfect. To achieve a better understanding of the problems that derive from this technology, we attempted to create our own self-driving vehicle based on a remote-controlled car. With information sourced by two cameras, it can calculate the proper behavior to navigate through an obstacle course made of pylons.**

***Keywords*—autonomous driving, automatization, software developement, student project**

I. Introduction

According to one definition, autonomous driving refers to fully automated driving of a vehicle without a driver interfering with the behavior of the car [1]. Automated driving is the process by which various assistance functions gradually control or intervene in the engine (acceleration), braking and steering systems. There are five levels of automated or autonomous driving defined by the SAE (Society of Automotive Engineers). All over the world, this classification has become the norm in the automotive industry.

Level 0: No automation. The driver steers, accelerates and brakes by himself.

Level 1: Automated systems such as antilock brakes (ABS) or electronic stability program (EPS) intervene automatically.

Level 2: Integrated systems take over partial tasks (e.g., adaptive cruise control, lane change assistant, automatic emergency braking). However, the driver retains control of the vehicle and responsibility.

Level 3: At certain points, the car can accelerate, brake and steer on its own (conditional automation). The system prompts the driver to take control if necessary.

Level 4: During normal operation, the vehicle can drive fully autonomously. However, the driver has the option to intervene and "override" the system.

Level 5: Completely automated, autonomous operation of the car without the possibility (and necessity) of intervention by the driver. [2]

Mercedes-Benz has been granted the world's first system approval for highly automated driving according to Level 3, which is valid internationally. The so-called "Drive Pilot" has been available in the premium S-Class model since 2022.[3] For example, drivers will be allowed to hand over responsibility to the system in traffic jams on certain Autobahn sections up to a speed of 60 km/h. However, drivers must remain "ready to take over."

Furthermore, autonomous driving is being tested on several routes in Germany - in cities such as Hamburg or Karlsruhe, but also in rural areas such as Upper Franconia or on company premises or campus areas. On these routes, self-driving shuttles will drive, which will be used in the future to transport people and goods.

Multiple universities from all over the world are offering students the possiblilty to work on vehicles for a race setting. As part of these Formula Student events, some are done with so called driverless vehicles.

Formula Student has been present in Germany since 2005 and has been attracting more and more racing teams every year. According to research, the driverless discipline of formula student has been around since 2016. In the future, according to the rules, every team must have an autonomous driving vehicle.[3]

The project of this paper has similar boundaries, just the car is of a smaller scale. The track we are given is limited with red/pink cones on one side and and blue/green cones on the other and our model is supposed to navigate through said course.

II. Approach

A. Motivation and Background

This project was a collaboration between the project team and HofSpannung, which is the Formula Student racing team of the University of applied Sciences Hof.[4]

Because of the shortage of computer science students that participate for Team Hofspannung, our Team decided to support their endeavours by trying to create a prototype plattform for them to work off.

To achieve this goal, the University supplied our Team with the necessary hardware. The base plattform is a remote-controlled car in the scale 1:10. The computational brain is a

Nvidia Jetson Development Kit [5]. The perception of the Environment is done with a ZED2 Camera [6] with stereoscopic lenses. Additionally, an I2C Servo converter was needed to control throttle and steering.

The basic idea is that the self-driving car drives through a course in which cones are used as boundaries on the left- and right-hand side of the track, as can be seen in Fig. 1.

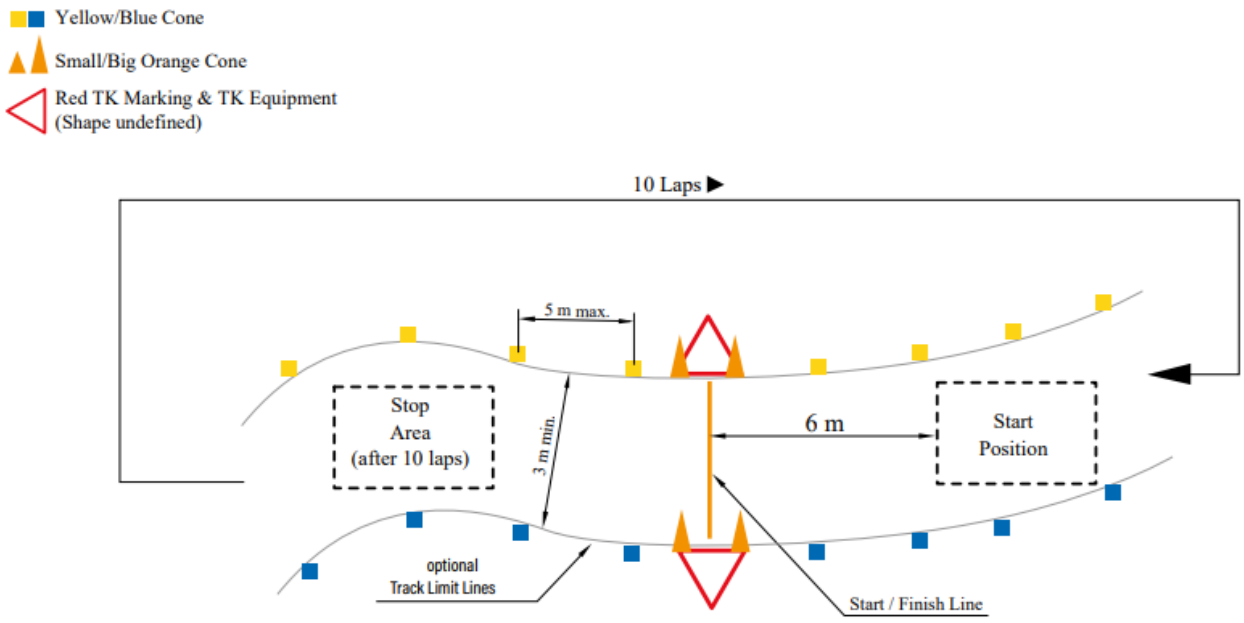


Figure 1: Course layout Formula Student

Therefore, the car must be able to recognize the cones while driving, calculate the trajetory and maneuver independently through the outlined course.

The project is split into several stages to accommodate for the almost insurmountable task at hand.

B. Stage 1: Taking control of the car

The first step is to familiarize with the car to achieve successful results. The preinstalled scripts from the previous car owner's group are used gain more insigth into the setup.

Following the successfull connection of the GPIO pins to the servo driver it is possible to control the car. The initial accleration test was perfomed on a mount mechanism, because the whole team was concerened about potential damage to the vehicle if the throttle was too high. This is all done through python scripts. To alleviate the control issues, the search for alternative control options for the interim period is done through a webinterface. This is especially useful to improve iteration times, since weblanguages are easy to change and adapt to the rapidly changing environment of the project's lifecycle.

A small Python web interface with two regulators, is used to steer and control the throttle. Albeit not perfect, the solution is feasible and easy to extend.

Alternative methods that were tested as well include an XBOX controller. The responsiveness is way superior to the web interface. However, it is difficult to control the car accurately because the input values and the power of the car's motor were not properly adjusted for.

C. Stage 2: Object Detection

After evaluating the preinstalled scripts and drivers on the Jetson, the team opted to do a clean install of NVIDIA's own Linux distribution called "Jetpack" [7]. This was mostly due to fact, that the Jetson platform has very specific driver recommendations and requirements. Positive sideeffects of the fresh install are an increase in performance as well as the possibility to use the CUDA cores for the needed model computation.

The ability to detect pylons is granted through the "YoloV5" [8] framework. To achieve this, a custom model is needed, since the supplied (self bought) pylons do not have an accessible model yet.

Initially hand-crafting training data on the Roboflow platform [9] was deemed the best solution. But the class loss and overall quality of the system was not sufficient enough. The final model is using a procedural approach to generate huge amounts of training data.

A script ("imagemerger") [10] made by a member of HofSpannung can generate a multitude of images by layering raw pylon photos into random background pictures with different lighting/orientation. To put the training data into perspective, the old model uses 160 Images, whilst the procedural approach generates over 24000 pictures if necessary.

Having more training data is not always the issue. The main culprit of bad recognition in this case is caused by differences in image capture devices. To circumvent this, the raw image data for the cones and the backgrounds are gathered by using the ZED Camera instead of Smartphones. Because that is the simplest way to have coherent input while driving regarding the training data. Fig. 2 is showing the working pylon detection.

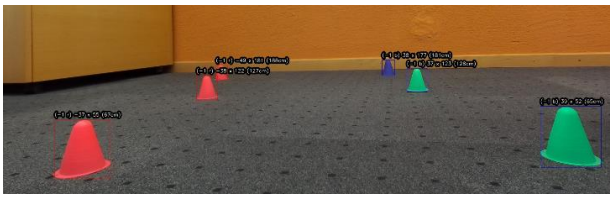


Figure 2: pylon detection on non-reflective university flooring

D. Stage 3: Coordinate System

The successful detection of objects is leading to the next important step towards autonomy: locality detemination. For the vehicle to be able to drive anywhere, the location of itself and the target point need to be known variables. The ZED Camera has a feature called positional tracking [11], which means it can not only measure its location in world space relative to its surroundings but can also measure the distance to and location of the detected objects. This feature is crucial to be able to circumvent using a LIDAR scanner for distance measurements, which would increase the complexity of the system. Furthermore, the RC-Car platform has no space for an addional LIDAR scanner in a useful position.

The positional tracking works with the coupling of an inertia sensor and the depth sensing capabilities of the stereo camera setup, much like the human eye perceives depth information and the inner ear registers gravity and linear acceleration.

E. Stage 4: Path calculation

With successful point detection and positional tracking of objects, it is possible to calculate points for the vehicle to drive towards. The approach makes use of triangles "drawn" between opposite Pylons, e.g., two inner pylons and one outer pylon. The halfway points between opposite points make up a simple path between the boundaries of the track. The car is then using the closest of those points as the next target. After getting sufficiently close, it deletes that point from the "points not yet reached memory" and drives towards the next nearest point. This results in predictable and save driving behaviour. Fig. 3 shows the web interface displaying the detected points(red/blue) and the calculated driving path(white) from a standstill position, with the green dot referencing the cars position and orientation.

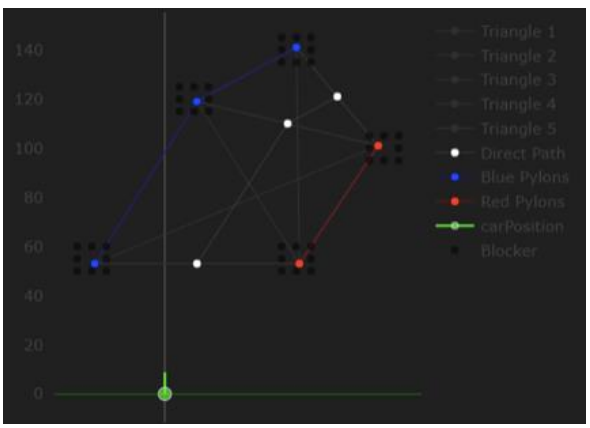


Figure 3: route calculation

F. Stage 5: Driving

After aquiring the points for the vehicle to drive to, the final step is to send these to a "control" mechanism which calculates the appropriate throttle and steering information to the servo controller. To avoid damaging the vehicle by applying incorrect velocity data to the car, it always receives 15% of maximum throttle. Steering data is then extrapolated by calculating the angle between the forward vector of the car and the vector towards the target point. The car then keeps steering until the two vectors are coincident.

G. Problemsolving

During the implementation multiple issues arose that needed to be fixed.

Initially the computational power of the jetson was unable to fullfill the requirements. Thus, a client/server system was created to distribute the workload more evenly. The ZED api allows for an unfiltered media stream to another network device. This is used to outsource the object detection to a more powerful computer equipped with a Gefore 3090, causing an FPS increase from <10 FPS to ~40 FPS. The importance of a high framerate comes from inaccurate measurements through vibrations and camera shake, thus needing multiple pictures per frame to verify pylon locations. An additonal LIDAR scanner could be used to verify the pylon position faster and more reliably. The remote system then sends the positional data back to the car. However, the seperation of computational workload is negativly affecting the performance, when the used network is congested, or the speed is inadequate.

Furthermore, multiple algorithms were constantly updated and optimized to increase performance.

H. Future steps

The next step would be to integrate the yellow cones as a start/stop or break area, as mentioned in Fig. 1(orange cones). Additionally, it was planned that the car would drive a lap and map the whole track, quickly calculate the optimal driving lines to achieve the fastest lap. Originally, we had also planned to integrate different disciplines, which can be viewed in the Formula Student rulebook [12]

III. Experiments

Over the course of the project multiple experiments were conducted to verify theories and get intermediate results.

A. Initial car tests

During the starting weeks the team got aquainted with the hardware and used the vehicle to perform some simple remote-controlled driving. This lead to an understanding of how dangerous it could become when the car gets out of control and what kind of behaviour to expect when given certain inputs.

B. Pylon recognition

As shown in Fig. 2, the object detection stage had us conduct multiple experiments in which we had to verify whether or not the model was capable of producing useful results.

C. *Driving test*

The first fully autonomous driving tests showed us, that the system had several flaws. The system relies on multiple precise measurements and verifications to determine valid course plotting. Because of many minor inconsistencies regarding lighting and distance measurement, the team found out that there is still a lot of refinement to do before fully autonomous driving on a pylon track will be possible. Fig. 4 shows, that there are too many pylons incorrectly detected.

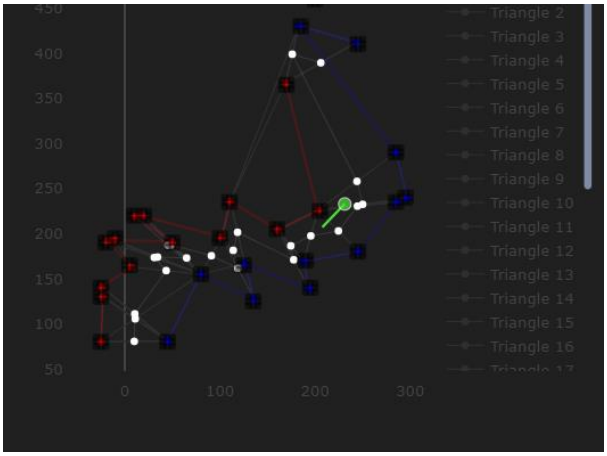


Figure 4: inconsistent pylon measuring

D. *Final Driving test*

Since there was no time to retrain the model from the ground up, the team conducted one final test. A manual set of points was sent to the car over the webinterface. The result was that the car continuously drove to the same final point no matter how often the test was repeated. The conclusion of this is, that the accuracy of the positional tracking of the car was very high and that the issue solely lies in the positional data of the pylons being inaccurate.

IV. Conclusion and Future Work

In the current state of the software, the vehicle can almost run autonomously through a parkour of cones. It receives information about the positions of the cones and uses this to calculate where it can drive. According to the calculation, the behavior of the vehicle adapts, i.e., it calculates the angle of the wheel position, as well as its own, relative position to the cones.

Level 4 autonomy is (partly) achieved. The vehicle is fully automated, and it is not necessary to intervene in the driving behavior. However, it lacks the precision in some areas to fully satisfy the expectations of the team.

In the future, this could be built upon by gradually introducing more actors and influences the vehicle needs to react to. However, the whole thing is limited, as is currently the case with Tesla, due to the structure of the software as a decision-making machine. To achieve true autonomy, an intelligent control system is required, which does not just react to influences in a predetermined manner but has situational awareness.

References

[1] https://de.wikipedia.org/wiki/Selbstfahrendes_Kraftfahrzeug

[2] https://www.adac.de/rund-ums-fahrzeug/ausstattung-technik-zubehoer/autonomes-fahren/grundlagen/autonomes-fahren-5-stufen/

[3] https://www.formulastudent.de/pr/news/details/article/rules-2023-v11-published/

[4] https://hofspannung.de/

[5] https://developer.nvidia.com/embedded/jetson-developer-kits

[6] https://www.stereolabs.com/zed-2/

[7] https://developer.nvidia.com/embedded/jetpack

[8] https://github.com/ultralytics/yolov5

[9] https://roboflow.com/

[10] https://github.com/timoxd7/ImageMerger

[11] https://www.stereolabs.com/docs/positional-tracking/

[12] https://www.formulastudent.de/pr/news/details/article/rules-2023-v11-published/

Disclaimer: No AI was used in the creation of this document.

Neural Flappy Bird using EMOTIV

Sayali Patukale
HOF University of Applied Sciences
HOF, Germany
sayali.patukale@hof-university.de

Shaunak Joshi
HOF University of Applied Sciences
HOF, Germany
shaunak.joshi@hof-university.de

Shreya Daga
HOF University of Applied Sciences
HOF, Germany
shreya.daga@hof-university.de

Sankalp Chordia
HOF University of Applied Sciences
HOF, Germany
sankalp.chordia@hof-university.de

Parth Vaidya
HOF University of Applied Sciences
HOF, Germany
parth.vaidya@hof-university.de

***Abstract*—The electroencephalogram (EEG) is the test that uses a multitude of tiny metal discs (electrodes) connected to the scalp to assess the electrical activity in the brain. Brain cells interact via electrical impulses and are constantly active, even while sleeping. On an EEG recording, this activity appears as wavy lines. This work aims to create a Brain-Computer Interface (BCI) solution that will allow a user to operate a Flappy Bird game using just his/her thoughts, which means a live and unprocessed stream of electroencephalogram data. We employ the Emotiv EPOC X, a commercially available EEG-based BCI device, for this experiment. The apparatus features 14 channels, which is more than adequate for this design (i.e., 14 electrodes to pick up impulses from the surface of the skull).**

***Index Terms*—EEG, EMOTIV, Game development, Flappy bird, BCI, HCI, Python**

I. Introduction

The brain is the most intricate system yet discovered, with 14-16 billion neurons. The EEG signal released by the human brain is a mash-up of all ideas of activity, moods, and other feelings [1]. Electrical signals are generated by ion movements passing through brain neurons, which are captured by EEGs. Neuronal activity generates circulating ion movements, and the space-time potential created by these ion movements is collected by putting electrodes over the scalp. Hans Berger recorded the first electrical EEG signal from the human brain using a d'Arsonval meter put in the brain with one electrode in the early twentieth century and found a single wave known as an alpha wave.

However, these systems are quite expensive and can require specialized training. Advanced EEG-based instrumentation systems are often based on numerous electrodes and can distinguish five basic waves, such as delta, theta, alpha, beta, and gamma.

An apparatus known as a brain-computer interface (BCI) transforms neural impulses from the cerebral region of the brain into information that may control external applications or hardware, such as a CPU or a robotic limb. BCIs are often used by persons with motor or sensory limitations as aids to daily life.

BCIs can be classified into two types: invasive and non-invasive. In the case of invasive BCIs, the subject's brain is implanted with electrodes or is stuck on the surface of the skull to get readings. But when compared to non-invasive ones, the electrodes are placed on the scalp of the user or placed along the surface of the head to get the data. The non-invasion method doesn't require any kind of surgery whereas the invasive does require a user to surgically implant the electrodes into his/her brain.

With the help of this method, a system that is more autonomous and less reliant on people will be introduced. These cutting-edge, cyber-physical system-based strategies will significantly alter educational, technological, and entertainment-based applications while using Brain signals.

II. Literature Review

Analysis of brain activity, the stimulus of nerve tissue, developments in technological advances, and robotic research enable interfacing between the human brain and artificial devices, known as Brain-Computer Interfaces (BCI). [1] Neural activity is detected by EEG as faint electrical potentials on the surface of the human scalp. The EEG signal is collected by applying electrodes to the forehead in the 10-20 system. However, EEG has limited spatial resolution (in the centimeter range), therefore it is difficult to match underlying neural activity with a single recording channel signal. However, in order to be recorded, the signal must pass through the skull and scalp; also, noise from the individual (eye motion, muscular activity), as well as external disturbances, are added (electronic devices, power line noise). [2]

There have previously been past efforts in this line of research where one of the projects regulated brain rhythms. The individuals were taught for approximately ten hours, and the results demonstrated that it is feasible to control left or right motion in a three-dimensional video game using brain impulses. [3]

In reality, whether invasive or non-invasive, the speed and accuracy of most modern BCIs are still significantly lower than those of systems based on eye movements. The majority of existing EEG-based BCI devices have an information transfer rate (ITR) of less than 0.5 bps.[5]

We hope to create a BCI-based flappy bird game in which the player may control the bird via neural activity.

III. Setup Overview

EPOC X is the most recent version of an electroencephalogram (EEG) analysis wireless headset that has been updated in response to input from the EMOTIV community. 14 EEG sensors and two matching sensors are used to detect brain waves for the development of brain wave applications employing neurofeedback.

Fig. 1. Emotiv EPOC X Headset [9]

The swivel headband allows it to be worn on either the upper or occipital area. The swivel headband helps you to gather more data in a more comfortable and precise manner, even while utilizing a headrest, such as while sleeping. Raw EEG data, as well as facial expression, emotional state, and mental state information, may be obtained using the program EmotivPRO. In this paper, we detect the brain activity of the user and accordingly make the bird move up or down.

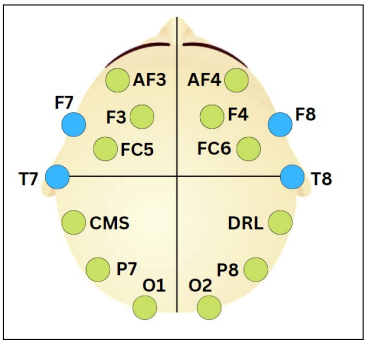


Fig. 2. Electrode Positions in the headset

The electrodes in such headsets can be broadly classified into three types: saline electrodes, needle electrodes, and cup-based electrodes. Silver chloride is the most commonly used for scalp electrodes. While attaching the electrodes to the scalp via the EMOTIV EPOC X headset, a saline solution is used. The solution is required so as to lower the impedance between the cortex and the electrode. To reduce aberrations in the EEG signal, we must use very low electrode impedance, which can be feasible according to our project. The precision of the brain wave signals is directly proportional to the wetness of the sponge electrodes that absorb the saline solution, the signals become less precise as soon as the saline solution evaporates and the electrode sponges start drying. To get the best results using this headset, we must precisely position the electrodes as stated in the user manual of the headset.

A. EMOTIV BCI Software Setup

[10] defines black, red, orange, yellow, and green as ”no signal,” ”very bad signal,” ”poor signal,” ”fair signal,” and ”good signal,” respectively. To get a green or yellow color, adjust the electrode slightly while pressing firmly for 10 seconds or applying more saline solution. When the headset is turned on, the gyroscope will activate.

IV. Game Development

A. Model Design

The Flappy Bird game development is based on the idea that a bird flaps its wings to rise with each screen contact and to descend when not contacting the screen. Meanwhile, the player receives 1 point for each successful passage of the bird through the barrier. The bird that collides with the boundaries or the floor dies, and the game finishes in this manner. Attention and meditation waves are employed as input to increase the game's competitiveness and enjoyment.

The steps involves in making the Fallpy Bird game were as follows:

1) Subscribe to the mental commands data from the EPOC X headset and print it in a python terminal.
2) Make the Flappy Bird game from scratch that is able to recognize brain activity i.e. that is able to use the mental commands data from the headset to control the movement of the bird in the game.
   - Flappy Bird Design: Starting initially with a dot for the Flappy Bird. The obstacles would be rectangles. It will have a 2D design and GUI. A basic design would be implemented initially.
   - The design will evolve by adding a picture of the Flappy Bird and 3D elements.
3) Test the Flappy Bird game using all three use case scenarios of the bird flying up or being stable to avoid the beams.
4) Adding advanced graphics to the Flappy Bird game.

B. Graphics

A lot of elements were used in the UI design of this game. The graphics used in this game were designed manually. The bird's movements can be seen in the game as shown in Fig. 3, Fig. 4, Fig. 5. The bird, the poles, and the system UI, with foreground and background, were used and designed by ourselves.

V. Methodology

The implementation of this project has two major parts. The first one being training of the model using the Live Mental Commands. The second being the implementation of this model in Python. The second part is further divided into two parts; those being the front end and the back end. The front end corresponds to the Flappy Bird Game being shown on the screen while the implementation of the code. The back

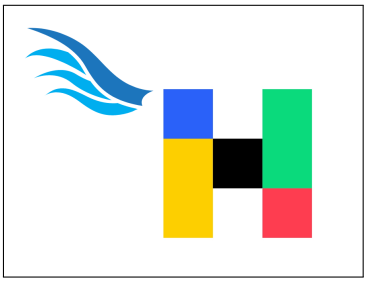


Fig. 3. Flappy Bird with it's wings up

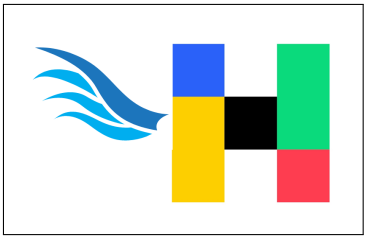


Fig. 4. Flappy Bird with it's wings in the middle

end is the connection to the Emotiv BCI to pull the action being performed and the strength of the action.

Training is a vital part of this project since the model needs to understand which action is being performed. Emotiv PRO offers the ability to train the model using its Mental Commands feature. Whenever a certain action is performed, there are small electrical impulses that are fired. This electrical activity is picked up by the EEG Headset. This is the basic working principle of any EEG Machine. Our model needs to understand which electrical activity corresponds to which action.

The headset needs to be connected properly before we train.

Emotiv PRO offers a number of different commands that can be trained in the application itself. These include PUSH, PULL, LIFT, DROP, etc. The application shows the user a block that you are supposed to perform the action on. The user is supposed to imagine themselves performing this action. A successful training iteration will lead to the block being moved according to the action. But before that, the user is supposed to train the neutral state of the brain. The neutral state is important as this is what the action commands are compared with.

For successful training of the model, a distinct graph is

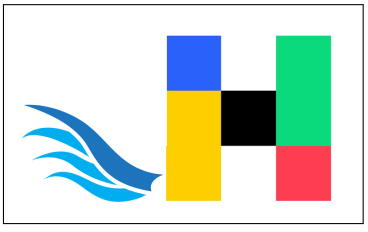


Fig. 5. Flappy Bird with it's wings down

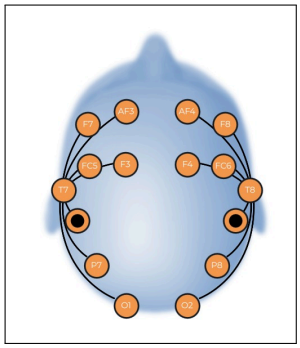


Fig. 6. Improper Connections

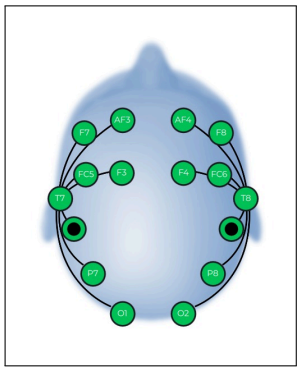


Fig. 7. Proper Connections

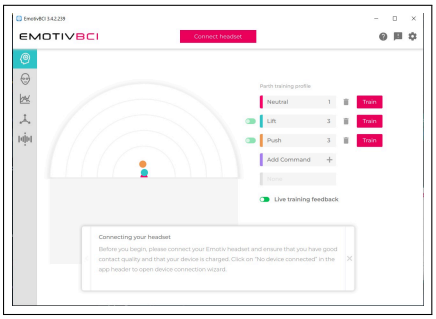


Fig. 8. Non Distinct Graph

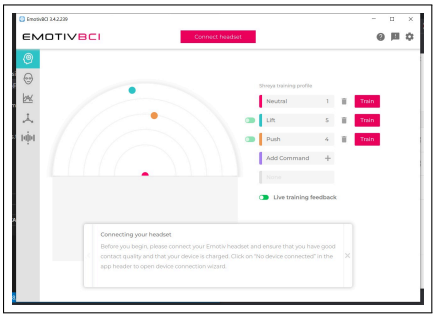


Fig. 9. Distinct Graph

necessary. With this graph, we can visually interpret and understand the training process. A distinct graph essentially means that the BCI model can distinguish between the brain wave states of ”neutral” and other mental commands such as ”lift” and ”push”. This helps the BCI model produce data that can be easily used to play the game. As you can see in Fig.8 the graph is not distinct as the distance between the dots representing the neutral state and mental commands is less. However, Fig.9 is a distinct graph as the distance between the dots is greater. Hence, the profile of the user depicted in Fig.9 would be most suitable for playing the game as the model will easily distinguish between mental commands and the neutral state.

After the training, we have two main parts to our project. The first is the code to run the Pygame interface. The second is the connection to the EPOC Emotiv BCI. This is done using the Cortex API. To use this we need to login into our specific Emotiv Account, so that it has access to the specific training profiles that we have trained.

The connection to the Emotiv BCI is done in further two parts. The back end is the Cortex API code which deals with the querying and the front end is the code that sets the sensitivity, handles the actions, and loads the profiles.

After the connection is done, the main criterion for the bird to jump is the power of the mental command. The power is denoted on a scale of 0.0 to 1.0, where 0.0 is the least and 1.0 is the most power. The power of the action (mental command) has to be more than 0.5 for the bird to jump. This includes any action, lift, push, pull, etc. Essentially, any brain activity will trigger the jump of the bird in the Pygame Interface. In our program, we have selected the mental commands ”lift” and ”push” to be monitored. If their power is above 0.5 the bird will jump (fly above).

We need to run the two parts of the program i.e. the front end and the back end of the game simultaneously. This is because we need to access the power of the action as well as run the game at the same time. This is done using the multiprocessing library in Python by setting up a server and a client. These two programs are run simultaneously in two different python terminals.

Fig 10 is a flowchart demonstrating how EMOTIV connects to the Python client and briefly explaining how the game functions.

VI. Results and Discussions

We have tested the application using a simulated headset. We have also tested the application with a human gamer operating the headset. The gamer was able to sit motionlessly and operate the game using just their brainwaves. Since we have used mental commands in our project, physical signals such as blinking or stress in the muscles are not used to operate the video game. The gamer is able to control the bird using solely their brainwaves. The results demonstrated that the users could play the game without moving any body muscles.

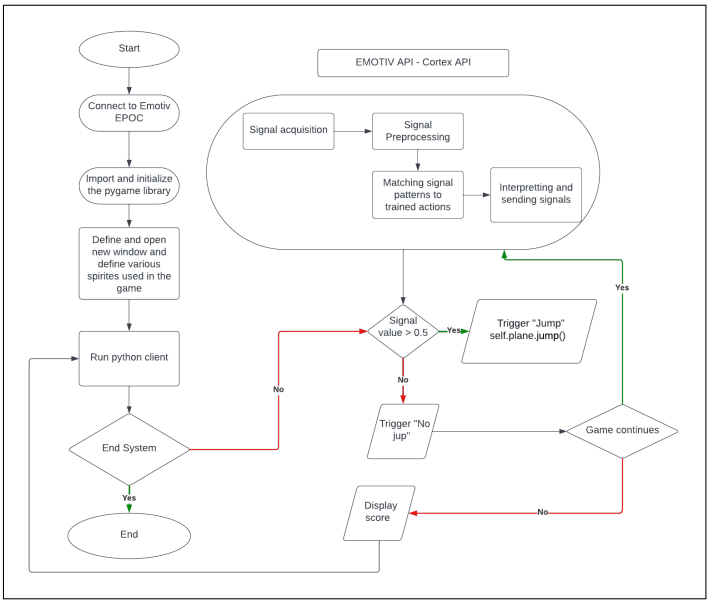


Fig. 10. Workflow of the system

Fig. 11. Starting Graphics of the Game

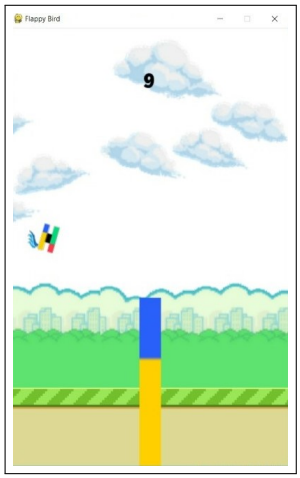


Fig. 12. While Playing the Game

VII. Conclusion and Future Scope

The applications of this project could benefit people with severe motor disabilities. An EEG system, for example, might produce commands directly from the brain to control one or more external devices for someone with a spinal cord injury. We can also expand this project further to make a Speech Generating Device. Initially, we can start with a game application and further, extend this application to controlling robotic arms using a brain-computer interface as the future scope of this project.

VIII. Acknowledgment

We would like to thank HOF University of Applied Sciences for giving us the opportunity and resources to work on this project. We hereby express our gratitude and thanks to our guide Prof. Christian Groth for providing his erudite guidance, vision support, and constant encouragement to complete our project successfully.

References

[1] Ali, A., Afridi, R., Soomro, T.A. et al. A Single-Channel Wireless EEG Headset Enabled Neural Activities Analysis for Mental Healthcare Applications. Wireless Pers Commun 125, 3699–3713 (2022). https://doi.org/10.1007/s11277-022-09731-w

[2] Kerous, B., Skola, F. and Liarokapis, F. EEG-based BCI and video games: a progress report. Virtual Reality 22, 119–135 (2018). https://doi.org/10.1007/s10055-017-0328-x

[3] J. A. Pineda, D. S. Silverman, A. Vankov and J. Hestenes, "Learning to control brain rhythms: making a brain-computer interface possible," in IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 11, no. 2, pp. 181-184, June 2003, doi: 10.1109/TNSRE.2003.814445.

[4] Varol, Asaf and Ulaş, Mustafa and YILDIRIM, NILAY. (2015). A Game Development for Android Devices Based on Brain Computer Interface Flying Brain. 77-81. 10.15224/978-1-63248-040-8-46.

[5] Zhao, Q., Zhang, L. and Cichocki, A. EEG-based asynchronous BCI control of a car in 3D virtual reality environments. Chin. Sci. Bull. 54, 78–87 (2009). https://doi.org/10.1007/s11434-008-0547-3

[6] Santhanam, G., Ryu, S., Yu, B. et al. A high-performance brain–computer interface. Nature 442,195–198 (2006). https://doi.org/10.1038/nature04968

[7] Muhammad N. Fakhruzzaman, Edwin Riksakomara, Hatma Suryotrisongko, EEG Wave Identification in Human Brain with Emotiv EPOC for Motor Imagery, Procedia Computer Science, Volume 72, 2015, Pages 269-276, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2015.12.140.

[8] R. Raju, C. Yang, C. Li and A. Cangelosi, "A video game design based on Emotiv Neuroheadset," 2016 International Conference on Advanced Robotics and Mechatronics (ICARM), Macau, China, 2016, pp. 14-19, doi: 10.1109/ICARM.2016.7606887

[9] Bharadwaj, Hemantha and Agarwal, Aayush and Chamola, Vinay and Lakkaniga, Rajiv and Hassija, Vikas and Guizani, Mohsen and Sikdar, Biplab. (2021). A Review on the Role of Machine Learning in Enabling IoT Based Healthcare Applications. IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3059858.

[10] K. Stytsenko, E. Jablonskis, and C. Prahm, "Evaluation of consumer eeg device emotiv epoc," in MEi: CogSci Conference 2011, Ljubljana, 2011.