Control Virtual Car With EEG Headset

Emre Agzikücük
Business Information Systems
Hof University of Applied Science
Hof.Germany
emre.agzikuecuek@hof-university.de

Semih Aktürk
Business Information Systems
Hof University of Applied Science
Hof.Germany
semih.aktuerk@hof-university.de

William Steve Kamguia Kammogne
Business Information Systems
Hof University of Applied Science
Hof.Germany
william.kamguia.kammogne@hof-university.de

Justus Maximilian Schubert
Business Information Systems
Hof University of Applied Science
Hof.Germany
justus.schubert@hof-university.de

Manuel Planer
Business Information Systems
Hof University of Applied Science
Hof.Germany
manuel.planer@hof-university.de

Abstract— **In this project, we used the Emotiv EpocX EEG headset in combination with the Unity engine to control a virtual vehicle by using the direct signals from the brain. While such interfaces have been explored in some other projects already, we wanted to tackle this concept through the development of a racing game experience. Although there is room for improvement in terms of precision and responsiveness, we view this as a promising step towards future applications of EEG technology.**

I. Introduction

In modern research scenarios, the connection between brain-computer interfaces (BCI) and virtual simulations is becoming increasingly important and crucial. That's why our challenge was to control a virtual vehicle using mental commands via an EEG headset (Emotiv EpocX). This challenge required the integration of various software and hardware components to efficiently process data and control the simulation environment. The main objective was to capture and process EEG signals to transform mental commands such as “move forward” or “steer” into actual vehicle movements within the simulation environment. This involved technical aspects of signal processing and user challenges for mental control.

Tools and technologies used:

- Emotiv EpocX EEG headset: Used to capture brain activity and convert it into control commands.
- Emotiv Cortex API: Used to access real-time data from the headset and integrate it into the control system.
- Unity: To simulate and visualize vehicle movements in a virtual urban environment.
- ROS (Robot Operating System) and Gazebo: initially used for robot simulation and control, they were later replaced by Unity due to its superior visualization capabilities.

The combination of these technologies has enabled us to explore innovative approaches to virtual vehicle control, and to evaluate the potential of EEG-based control mechanisms

II. Initial Approach with ROS

As part of the project an attempt was initially made to control the EEG headset with ROS in a Gazebo virtual environment. To begin with ROS 2 Humble was installed on an Ubuntu 22.04 Jellyfish as ROS Humble is only compatible with this version [1]. Gazebo was then set up as a visual environment and the Turtlebot 3 Gazebo package [2] was installed to simulate a moving vehicle. The Teleop Twist Keyboard [3] package was installed to enable initial attempts to control the TurtleBot robot using the keyboard. This was to form the basis for controlling the robot. An attempt was made to rewrite the keyboard package so that it works with EEG mental commands. Thanks to the Python scripts already published by Emotiv in their official Git repository, it was possible to adapt them quickly. The focus was on the already available scripts that allow access to the Cortex API and communication between the headset and the generated data. An attempt was also made to develop a ROS node that would access this data. However, after some time we realized that a software called Emotiv BCI was required to train Emotivs mental commands. Unfortunately, this was only compatible with the Windows and macOS operating systems at the time of the project. Although the Emotiv Launcher was compatible with Linux, we did not realize at the time that we would need the BCI software. As a result, we considered developing our own neural network to train the mental commands ourselves. However, this proved to be too extensive, as we did not yet know how the signals could be interpreted, and which signal was required to make precise predictions based on thought. We therefore questioned our previous decision. Furthermore, the possibilities in Gazebo were also limited. It does not offer the same visualization capabilities as other software, such as Unity or Unreal Engine, which can produce extremely high-quality visualizations.

III. Training Emotiv

A. *Description of the Emotiv EpocX EEG headset*

The Emotiv EpocX is an advanced 14-channel EEG headset designed for neuroscience and BCI applications that provides precise insights into brain activity. It is our central tool for the project and enables thoughts to be converted into control commands for our virtual car. The electrodes are strategically placed on the scalp to precisely record the electrical activity of the brain.

The electrodes work optimally if they are supplemented with small foam pads that are carefully soaked in a saline solution before use. This solution ensures that the conductivity between the skin and the electrodes is improved, which is essential for stable signal transmission. The pads are inserted into the contacts and prepared before each training session to ensure that the signals are reliably recorded. Experts recommend the use of salty liquids such as saline solution or contact lens solution, as these can significantly improve the connection. Tap water, on the other hand, is unsuitable as it does not sufficiently support the signal quality [4].

A USB dongle is used to connect the headset to a laptop. The associated Emotiv Launcher monitors and analyzes the recorded signals. The real-time display of the electrode contact quality is particularly useful for setting up the optimal position of the headset on the scalp. At the same time, the software offers various calibration and analysis tools that can be used for subsequent training.

B. *Problems during training*

Training with the Emotiv EpocX EEG headset presented us with a few challenges. One of the main difficulties was the calibration of the device on the user's head. The electrodes had to be precisely positioned, and the skin contacts optimally established to ensure a stable signal quality. It often occurred during testing that ambient noise or the user's own movements lead to a loss of signal and the headset had to be repositioned.

Another problem was the subjectivity of the mental states. Each user has individual thought patterns, and it was a challenge to focus thoughts on such a way that they could be reliably recognized by the system. Technical limitations, such as the limited accuracy in distinguishing between similar mental states, were also noticeable.

Another major problem was the different performance of the device among the team members. While the headset would work relatively stable and reliably for one person, some signals were consistently absent for others. This was probably due to individual differences, such as the condition of the scalp, hair structure, or even minimal variations in electrode placement. It was particularly frustrating when sufficient results could not be achieved after prolonged calibration.

A further obstacle also emerged: the headset's battery would drain rapidly. The device often had to be recharged after just one hour of intensive training, which regularly interrupted progress and made the already complex process even more difficult.

C. *Mind training*

An essential part of the training process was working with the Cube, a visual aid within Emotiv BCI that can be moved around with the trained mental commands. We used this tool to check the results of our training progress and try out different thought patterns with varying degrees of success. To make progress, we had to complete many training sessions and create a variety of profiles. Each profile represented a specific calibration, but finding a reliable pattern was challenging. Fine-tuning was particularly time-consuming, as the connection to the headset was often unstable and affected training results.

During the training phase, we experimented with a variety of thought concepts. These included concepts such as fluidity, emotions, temperatures, and concrete objects. The approach of linking physical sensations or movements to the control commands was particularly helpful. For example, we imagined how a liquid moves in the head or how temperatures are felt. Studies and guides to similar systems suggest that thoughts such as the taste of lemons or the sensation of dipping a hand in hot water often produce good results. It is also recommended to imagine physical movements, such as pushing an imaginary ball in a certain direction [5]. After many tests and adjustments, we finally succeeded in creating a profile that delivered stable results and served as the basis for our project.

In the final training profile, specific thoughts were linked to the individual control commands:

- **Driving forward:** I imagined pushing an object away from me with force.
- **Reversing**: I thought about pulling a rope towards me, like when hauling in a heavy object.
- **Drive left**: The thought of cold temperatures such as ice, snow, or cold water served as a trigger here.
- **Drive right:** The focus was on hot temperatures, such as the sun, fire, or the feeling of burning.

This approach was not only intuitive, but also consistent enough to be successfully applied in the subsequent test phases.

D. *Other relevant points*

Another interesting aspect was the learning curve. The precision improved with each training day, which indicates that both the system and the user learn from each other. Mental and physical preparation was also important: sufficient sleep and little stress significantly improved the signal quality.

IV. Making the Unity project

A. *Assets and Plugins used in the project*

Emotiv, the company behind the EpocX headset, released a Unity plugin that allows us access to the Cortex service. Inside, the script EmotivUnityItf.cs acts as an interface for us to access different methods of the plugin. By following these steps, we can get the streamed data from our headset into unity:

1. Establishing a session with the headset
2. Loading a training profile
3. Subscribing to the data stream

We then modified EmotivUnityItf.cs to create public variables from the streamed values. They enable us to check for the actual commands and their power from outside the script and then when we implement the car controls, we can use them to control its movement.

For the creation of the game, we also needed to incorporate 3D assets. Yet, we didn’t have enough time or 3D modelling experience to create them on our own. So, we decided to check the unity asset store and see whether there was anything available for free that we could use.

For the car we used “Polystang CC” by Faber Brizio, an asset pack with a controllable car and some road elements. By default, the car can be controlled via keyboard using WASD input and a camera can be set up to follow it around. This package also includes a speed meter UI element, engine sounds, lights and skid marks. The car controls are in Car control.cs where we later added the Headset as an additional control mode and configured its expected behavior.

For the environment of the game, we used "CITY package" by pixel studios. An asset containing different city themed objects like buildings, traffic cones, fences, etc. These objects provided our game with a more interesting backdrop than before.

B. *Building the game*

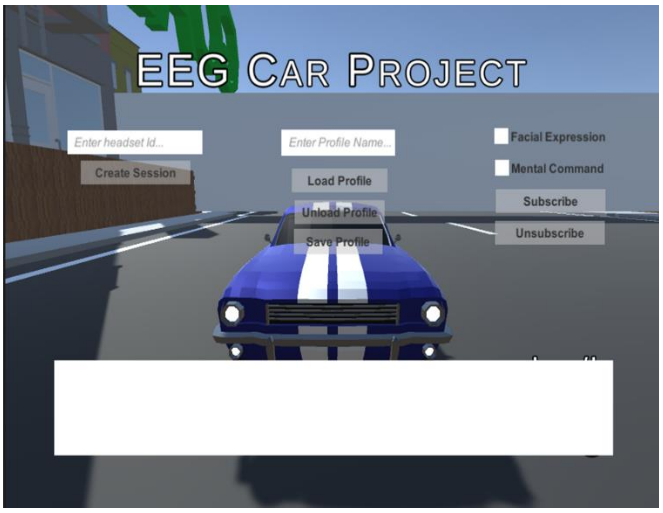
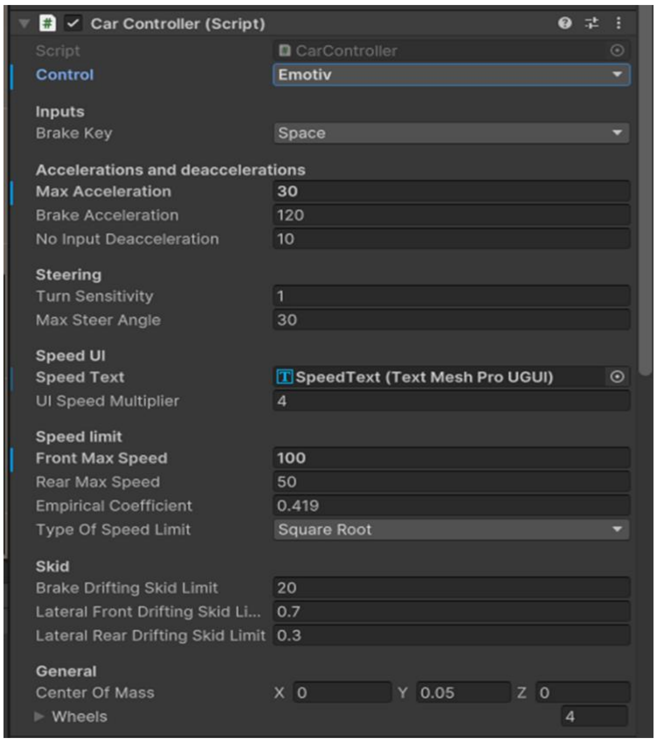


Fig. 1. Unity game start screen and Emotive user setup

After importing all the assets that we needed, we decided to first try and see whether we could get the mental command input from the headset into the unity console. This didn't prove too much of a challenge; we just needed a script to provide the ClientID and ClientSecretID to EmotivUnityItf.cs.
For the first tests we set up the car and a simple straight road segment and we also modified the car control script by adding the headset as a new control mode on top of the current keyboard and touch controls. The headset control mode can be selected in the inspector, and it is set up to automatically disable any keyboard input when used. By checking whether the mental command is active and what power it has, we were able to control the car without using the keyboard for the first time. Later we had to change the steering controls from simply "left" and "right" to "forward-left" and "forward-right" because having to steer and accelerate at the same time was not possible when only one command can come in at a time.

For the data stream we decided to use mental commands (push, pull, left, right) as it seemed to be the most intuitive. However, we also experimented with facial controls via different expressions. Both can be used depending on the selection in the set-up screen.

In many of our earlier attempts the car would swerve by accident and drive off the road leading to an endless fall that ended the game. So, we added a big ground plane to avoid repeatedly restarting the game due to minor input errors. We also noticed how even tiny changes in the mental command data would send the car off speeding in the opposite direction than where we wanted. To fix this, we tweaked some variables regarding top speed and acceleration in the unity inspector. This led to a more comfortable and responsive driving experience for the user.


Car Controller (Script)

Script	CarController
Control	Emotiv
Inputs	
Brake Key	Space
Accelerations and deaccelerations	
Max Acceleration	30
Brake Acceleration	120
No Input Deacceleration	10
Steering	
Turn Sensitivity	1
Max Steer Angle	30
Speed UI	
Speed Text	SpeedText (Text Mesh Pro UGUI)
UI Speed Multiplier	4
Speed limit	
Front Max Speed	100
Rear Max Speed	50
Empirical Coefficient	0.419
Type Of Speed Limit	Square Root
Skid	
Brake Drifting Skid Limit	20
Lateral Front Drifting Skid Li...	0.7
Lateral Rear Drifting Skid Limit	0.3
General	
Center Of Mass	X 0 Y 0.05 Z 0
Wheels	4

Fig. 2. Unity inspector changes to car behavior

Later, we set up a square shaped track according to our specifications with a finish line at the end. When reaching it the game would end with a winning message. In our experiments to reach the goal using only the EEG headset we discovered some problems. The first issue was that the car would often swerve off the road and end up getting stuck outside. We decided to surround the track with a fence to make the game easier to clear. However, now the car would get stuck on the fence, so we had to reduce the friction and increase the bounciness of the car by using a physics material with the appropriate settings.
Next, we defined the players goal further. But outside of reaching the finish line we had conflicting ideas. We decided in the end to go for a time trial approach where the goal would be to complete the game with as low of a time spent driving as possible. We added a timer, managed by Timer.cs, that would begin incrementing as soon as the data stream is subscribed to and stop when the car crosses the finish line. The collision with the finish line is handled within Finish.cs.

C. *Game Progression*

When starting the game, players are greeted with the initial headset setup user interface. Inspired by the simple example unity project that Emotiv made we also used text fields and buttons to manage the interaction with Cortex. These features require the user to be logged into the Cortex launcher at the same time. There they can also monitor the headset connectivity and signal stability during use.

In the set-up phase the user is prompted to interact with the UI from left to right. The Textbox at the bottom provide feedback to the different set up stages. First, they must create a session, inputting the headset ID for this is optional. Then, they load a training profile by typing in the name and clicking "load profile". Profiles should be premade by the user in the BCI launcher. Finally, they can choose their preferred control mode. We offer both facial expression and mental commands as they both meet the demands of our project. When the user clicks on "Subscribe" the data stream is ready and the game can begin. With that the initial set up screen vanishes, giving way to the game itself. This is also the point where the clock starts ticking as the timer begins to increase.

The game has a simple design. The player starts in one corner of the square shaped track and their task is to complete one loop around it. The timer stops when the finish line is crossed by the player, acting as a score to their individual performance. The lower the time spent navigating the track and its challenges, the better. A barrier at the start prevents cheating by driving backwards into the finish line.

The player must navigate to the finish line using only the Cortex Headset without using anything else. However, there are multiple obstacles in the way, one on each side of the square. We kept the obstacles simple because from our experience the unfamiliarity of using the headset adds to the challenge in many unexpected ways.

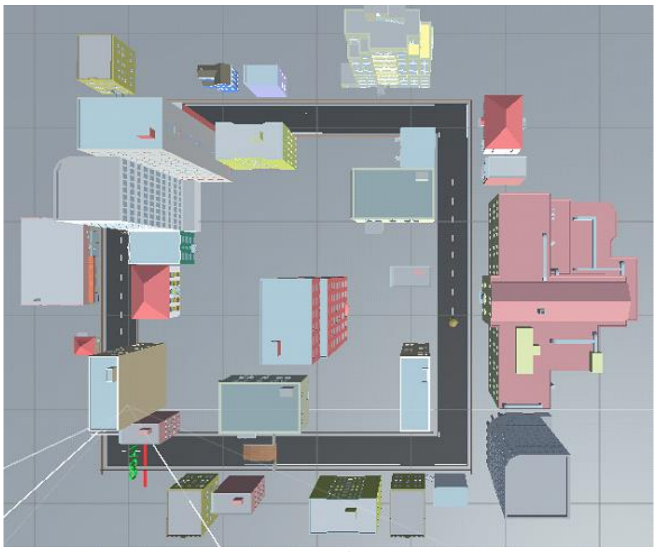


Fig. 3. Map of the track

D. *Cones on the Right side*

The first challenge involves orange traffic cones placed on the right side of the street. The car starts on the right side as well so simply driving forward is not good enough to clear this obstacle. The player is expected to navigate to the left in order to dodge.

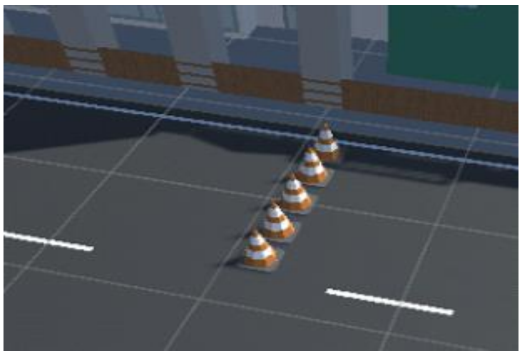


Fig. 4. obstacle with cones

E. *Fences on the Left side*

The second challenge is a row of construction fencing running along the left side of the track. Here the player should dodge to the right in order to avoid them. They also must turn right at each corner of the square, else they run into the fence barrier outside of the track. When the player ends up running into an obstacle or barrier by mistake, they must reverse the car and try again. The end of this side of the square marks the halfway point of the game.

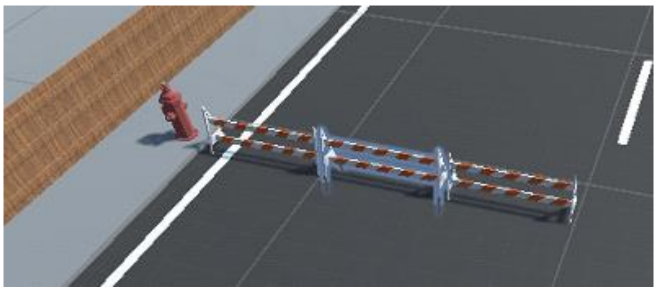


Fig. 5. barrier obstacle

F. *Object sitting in the middle*

This obstacle gives the player a choice, whether they want to turn left or right to dodge it.

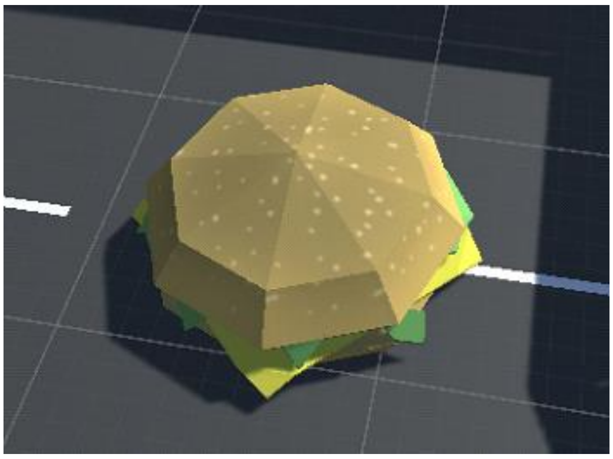


Fig. 6. Obstacle in the middle

G. *Obstacle 4: Ramp*

The final obstacle is a ramp located in the center of the track. Players must navigate this situation carefully adjusting their speed and angle before they drive onto it. Driving too fast or hitting the ramp at the wrong angle could cause the vehicle to lose balance or even flip over. This is the last obstacle of the game and clearing it leads directly to the finish line. When the end is reached the timer stops and a winning message pops up marking the end of the game.

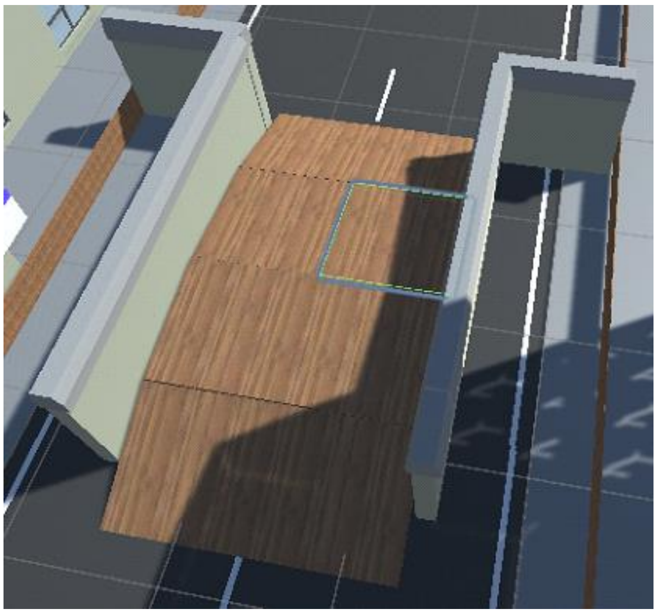


Fig. 7. final ramp for the goal.

V. Possible solution with ROS

After consideration, we came up with another approach to create an interface between ROS and BCI despite the operating system dependency of the BCI software. One possibility is to use the rosbridge_suite, which provides a webSocket interface and enables cross platform communication. [6] This allows a server to be set up that is able to receive data from the Emotiv BCI software in real time. This data can then be made available via the ROS Topics and subscribed to by the ROS nodes used to control the simulated robot, therefore ensuring that the data is transmitted with minimal latency and control via mental commands is possible.

VI. Conclusion

Due to technical problems caused by the limited compatibility of the Emotiv BCI software with Linux the project was completed in Unity instead. Unity provided the necessary support options, including a plugin from Emotiv that allowed the EEG headset signals to be received via the BCI software. As a result a working solution was developed.The control of the virtual vehicle was based on mental commands that had been individually trained using mental imagery in the BCI software. For example, imagining the action of pushing an object away made the vehicle move forward, while steering left was linked to the mental association of cold. One of the main challenges was calibrating the EEG headset, as factors like the user's hair structure, signal quality, and ability to concentrate had a big influence on the data and signals. It was especially important to position the EEG electrodes correctly to ensure a stable and reliable signal, so the intended commands could be accurately used during training.At the same time a virtual track for the vehicle was created in Unity. Different Unity assets, such as a vehicle model and various buildings from the Unity Asset Store, were used to build a course. The control logic included functions like moving forward and backward, as well as steering right and left, to give full control over the vehicle.The project demonstrates how vehicles could potentially be controlled by mental commands in the future.

Future Perspectives:

Potential future improvements include:

- Improved signal processing using advanced neural networks for more precise interpretation of mental commands.
- Use of Unity to create more realistic environments and enhance the user experience.
- Exploration of additional control mechanisms such as facial expressions combined with mental commands.

This project demonstrated both the potential and limitations of current EEG-based control systems, laying the groundwork for further developments in this field of research.

References

[1] ROS, "ROS 2 Documentation Ubuntu,"Available: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html (accessed Dec. 14 ,2024)

[2] Robotis, "TurtleBot" Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/ (a (accessed Dec. 14 ,2024)

[3] Graylin Trevor Jay, "teleop twist keyboard".Available: https://wiki.ros.org/teleop_twist_keyboard (accessed Dec. 14 ,2024)

[4] Emotiv, "Adaptive control for singularly perturbed systems examples,"Available:https://www.emotiv.com/tools/knowledge-base/general-issues/tips-for-getting-good-contact-quality-with-the-emotiv-headset (accessed Dec. 2 ,2024)

[5] Emotiv, "Tips for Getting Good Contact Quality with the Emotiv Headset"Available:https://www.emotiv.com/tools/knowledge-base/emotivbci/tips-on-mental-command-training?srsltid=AfmBOopwLY6r7dy9FZAsY-IkTemaXAAtW6FWJ6a2qSIwLwPzF1GIW1Zt, (accessed Dec. 5 ,2024)

[6] Jonathan Mace, "Rosbridge suite," Available: https://wiki.ros.org/rosbridge_suite (accessed Jan.5,2025)

conference paper prior to submission to the conference.
Failure to remove template text from your paper
may result in your paper not being published.

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi TIFF or EPS file, with all fonts embedded) because,in an MSW document, this method is somewhat more stable han directly inserting a picture.

To have non-visible rules on your frame, use the MSWord “Format” pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.