OSLO METROPOLITAN UNIVERSITY
STORBYUNIVERSITETET

Robot Assisted Deep Brain Implantation

Adam Emile Aske
Faten Alkhair
Yang Solås

Department of Biomedical Engineering
Oslo Metropolitan University

ACIT4720: Medical Sensors and Actuators

13.11.2023

Contents

Table of Figures

Acknowledgements

Thank you to Knut-Olav Utistog for his generous contribution of 3D models of medical equipment.

Introduction

Millions of patients across the world suffer from epileptic seizures. Many patient's seizures are severely damaging their quality of life. Using Stereo-Electroencephalography (SEEG) surgeons can implant electrodes deep into a patient's brain and record its activity pre, during and post seizures. Analyzing the data can reveal insights into the nature of the seizures and the patient's treatment can be adjusted accordingly. This report documents the authors planning, implementation and testing of a novel system for sensory-enabled robot assisted deep brain implantation. The system is hereby referred to as both "this project" or "this system".

In this project, a system for robot assisted deep brain implantation was developed. The utilization of robotic assistance in invasive surgeries presents numerous advantages, such as enhanced reliability, speed, and safety. The Braccio Tinkerkit Robot was used together with an Arduino Uno and a computer. To assist development, a simulation has been developed. It involves the application of robot kinematics to calculate the angles and positions of the robotic arm, the use of a gyroscope to detect angles, 3D printing technology to replicate the head, the base for the head, and the modification of the robotic arm's end-effector. The communication aspect is facilitated through the implementation of the Arduino. Ultimately, by employing software tools like Python and Unreal Engine 5, the project successfully achieves the simulation of Robot-Assisted Deep Brain Implantation.

Background

Epilepsy

65 million people worldwide (Patel, 2019) and 32500 people in Norway are diagnosed with Epilepsy (Dahl-Hansen, 2019). Epileptic seizures result from irregular activity of brain cells called neurons. Neurosurgery may be needed when medicines do not control or alleviate seizures. The type of surgery needed depends on the location of the neurons that start the seizure and the age of the person having the surgery. All types of epilepsy surgery must pinpoint a small portion of brain tissue. Therefore, there are some dangerous risks with such surgery. The risks can include damaging brain regions responsible for memory, language, visual and others.

Robot Assisted Deep Brain Implantation

As the name suggests, robot assisted deep brain implantation (RADBI), involves using robots as an assistive tool for neurosurgery. Robots as surgical assistants is an emerging technology. Using robots has both advantages and disadvantages, discussed later in the discussion section. Further development of robot assisted deep brain implantation technology has the potential to increase reliability, speed and safety of these surgeries.

Deeb Brain Stimulation

Deep Brain Stimulation (DBS) is a commonly used paradigm for neurological treatments. Some DBS systems use noninvasive techniques. Invasive techniques often involve implanting electrodes into the brain of the patient.

Robot Kinematics

In robotics, kinematic calculations are often done in matrix form. Matrices allow for simplified expressions and computers can very efficiently calculate matrix multiplications. Once the proper matrices are defined, the location of the robot's end effector can be calculated by a series of matrix multiplications. Calculating the location of the end-effector in relation the robot's base is the most common use case for this branch of mathematics. Before matrices can be defined, frames are defined. Figure 1 depicts an example of two frames, a fixed and a moving frame, and points P, Q and R.

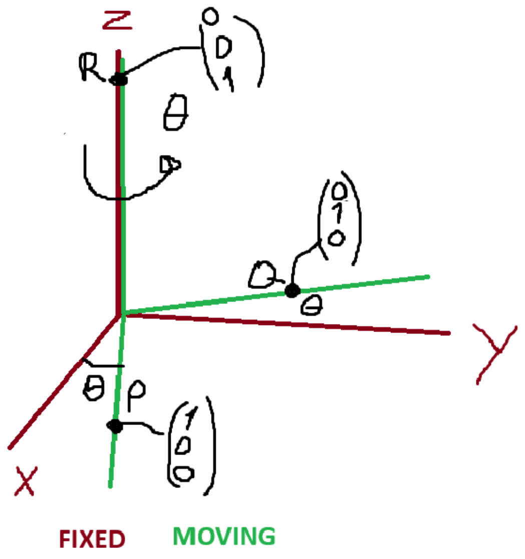


Figure 1 A fixed frame and a moving frame

The translation vector is a set of values defining a point's translation in space from a frame's origin. Notice from Figure 1 that P, Q and R, are translation vectors. Based on Figure 1, this means P is translated 1 unit on the x-axis in the moving frame. Now, what is P's translation in the fixed frame? We can see from Figure 1 that rotating the moving frame by $-\theta$ on the z-axis aligns both frames. With aligned frames, P's translation vector is the same for both frames. Thus, having P's translation in one frame, its translation in another frame can calculated by accounting the rotational relationship of the frames. The translation vector is defined by:

$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Where, $x, y\ and\ z$ are the translations, respectively along the x, y and z axes.

The rotation matrix is defined by:

$$R = \begin{pmatrix} R_{00} & R_{01} & R_{02} \\ R_{10} & R_{11} & R_{12} \\ R_{20} & R_{21} & R_{22} \end{pmatrix}$$

Again, using the example from Figure 1:

$$X = A * P = \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix} * \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Where X is point P’s translation vector in the fixed frame and A is a rotation matrix. This means that point P multiplied with rotation matrix A equals P’s translation in the fixed frame. This equation can be solved using the fact that the translation vector will be the same if the frames are aligned. Solving the equation yields:

$$A = \begin{pmatrix} \cos(\theta) & ? & ? \\ \sin(\theta) & ? & ? \\ 0 & ? & ? \end{pmatrix}$$

Applying the same technique for points Q and R as well, results in:

$$A = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Thus, A the z-axis rotation matrix of the moving frame. Repeating this with the moving frame rotated along the x and y axis respectively, rotation matrices for all axes are defined:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Where $R_x(\theta)$ describes a rotation of θ degrees along the x-axis (x, y and z).

The transformation matrix is defined by,

$$T = \begin{pmatrix} R & P \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_{00} & R_{01} & R_{02} & P_x \\ R_{10} & R_{11} & R_{12} & P_y \\ R_{20} & R_{21} & R_{22} & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Where, R is the rotation matrix and P is the translation vector. The last row in the matrix defines it as a homogenous transformation matrix with a scale-factor of 1. With these matrices define, if all frames are known, any point's translation vector can be calculated for any frame. Luckily, robots already have defined frames and translation vectors, thus the robot kinematics can be calculated.

Unreal Engine and Simulation

Unreal Engine 5 (UE5) is traditionally a game-engine. However, it carries usefulness far beyond the scope of video games. Due to the nature of modern games, game-engines include a variety of technologies from physics to rendering to user-interface and beyond.

Methods

Preproduction

After defining the goal of the project, the authors conducted a literature review to research existing systems and solutions for RADBI. A combination of keywords: "Deep brain implantation", "Deep brain stimulation", "Robot assisted" and "Robot", were used with Google Scholar to find sources. From the review, the authors identified spatial information between brain and robot, as the biggest challenge for this project. Hence, research into real-time head-tracking was done. The solution to this challenge is described later in this section. A specification requirement sheet was also created for an accelerometer, gyroscope, pressure sensor and IR sensor, but only the accelerometer and gyroscope were used.

The Robot

A Braccio Tinkerkit Robot Arm is used for the robot arm. The robot is connected to an Arduino Uno R3, stacked with a Braccio Shield module. Figure 2 shows the Braccio robot. The goal of the robot is to position an insertion tool accurately onto a patient's (3D printed head, discussed later) scalp. The end-effector of the robot is the white gripper. The gripper does not suit the goal; therefore a custom end-effector was made.

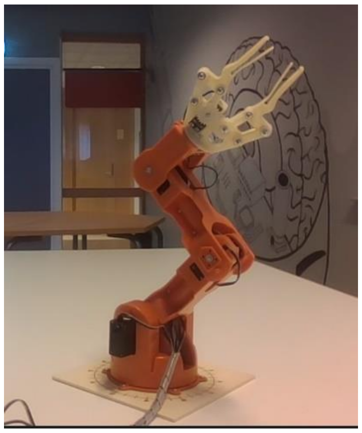


Figure 2 Braccio Tinkerkit Arduino

Through the design process for the end-effector, there were different elements to consider. The end-effector had been suitable with the robotic arm, light, simple and easy to assemble and disassemble. While the tool should be interchangeable to suit the requirements of the surgery. The tool was modeled, shown in Figure 3. 3D printing was chosen to produce the end-effector. The available material was PLA, but chromium was recommended to produce the insertion tool.

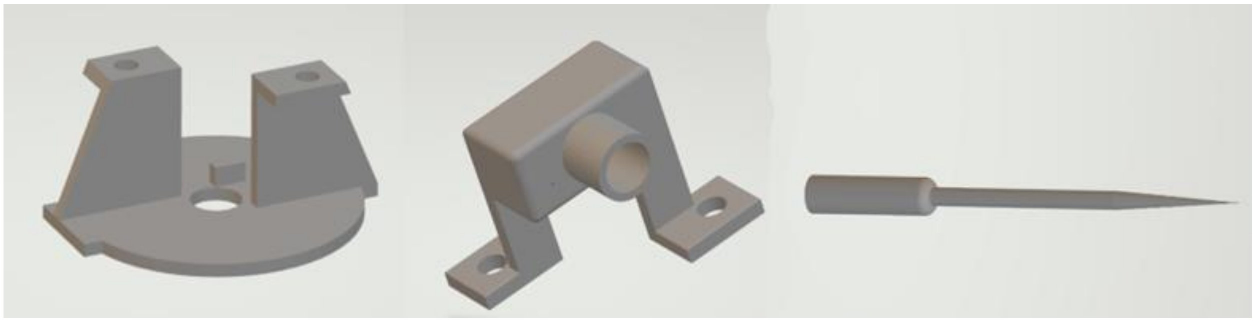


Figure 3Custom end-effector and insertion tool.

Gyroscope

Accuracy – Double Checks – Insight into robot position errors - MPU6050 Gyroscope and Accelerometer – The arm neither moves nor rotates fast, therefore the specified requirements for the Gyroscope and Accelerometer are low.

Specification requirements –

To use it we need it to output its current degrees, not how much it is currently moving. This resulted in a calibration function.

Robot Kinematics

Forward kinematics is the method of calculating the pose of the robot using the angles of each motor and length of each link. Inverse kinematics is the opposite, finding the desired angles for the robot to position of the end-effector at the desired location. Both forward and inverse kinematics was calculated.

Using MATLAB, a symbolic representation of the inverse kinematic was generated, shown in Figure 4. To numerically calculate the inverse kinematics equations from this large equation, is and was practically impossible for the authors. This is a common situation in robot kinematics, the more variables the robot functions from, the more complex equations are needed.

$$\begin{pmatrix} c_4\,\sigma_3 - s_0\,s_4 & -s_4\,\sigma_3 - c_4\,s_0 & \sigma_4 & d_4\,\sigma_4 + c_0\,d_2\,s_1 + c_0\,d_3\,\sigma_6 \\ c_0\,s_4 - c_4\,\sigma_1 & c_0\,c_4 + s_4\,\sigma_1 & \sigma_2 & d_4\,\sigma_2 + d_3\,s_0\,\sigma_6 + d_2\,s_0\,s_1 \\ -c_3\,c_4\,\sigma_6 - c_4\,s_3\,\sigma_7 & s_3\,s_4\,\sigma_7 + c_3\,s_4\,\sigma_6 & \sigma_5 & d_0 + d_1 + c_1\,d_2 + d_4\,\sigma_5 + d_3\,\sigma_7 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where

$$\sigma_1 = s_0\,s_3\,\sigma_6 - c_3\,s_0\,\sigma_7$$

$$\sigma_2 = s_0\,s_3\,\sigma_7 + c_3\,s_0\,\sigma_6$$

$$\sigma_3 = c_0\,c_3\,\sigma_7 - c_0\,s_3\,\sigma_6$$

$$\sigma_4 = c_0\,c_3\,\sigma_6 + c_0\,s_3\,\sigma_7$$

$$\sigma_5 = c_3\,\sigma_7 - s_3\,\sigma_6$$

$$\sigma_6 = c_1\,s_2 + c_2\,s_1$$

$$\sigma_7 = c_1\,c_2 - s_1\,s_2$$

Figure 4 Image of IK function for robot arm.

To solve the IK equations, the equation must be simplified. The Denavit-Harternberg (DH) method is a common method to accomplish the simplification. The DH method introduced mathematical constraints on the robot. The constraints are defined:

1. The x_n-axis must be perpendicular to both the z_{n-1} and z_n axes.
2. The x_n-axis intersects both the z_{n-1} and z_n axes.
3. The origin of joint n is the intersection of x_n and z_n.
4. The axis y_n must be aligned to complete a right-handed reference frame based on x_n and z_n.

Where n denotes the index of the link.

A DH model is defined by a set of 4 parameters per link in the robotic chain. θ represents the rotation along the z-axis (degrees). α is the rotation along the x-axis (degrees). D and r represents the translation along the z and x axes respectively. Adhering to the constraints the DH parameters were calculated. Tabell 1 shows the parameters for each link.

$Link$	θ	α	d	r
0	θ_0	90	7.15	0

1	θ_1	0	0	12.5
2	θ_2	0	0	12.5
3	θ_3	0	0	12.5
4	θ_4	0	0	4.8

Tabell 1 Denavit-Harternberg parameters.

Where θ_n is the rotation along the z-axis at joint index n. In Kin_Math.py, referenced earlier, each row in the able is realized in code as shown in Figure 5. Using Tabell 1, both forward and inverse kinematics can be calculated.

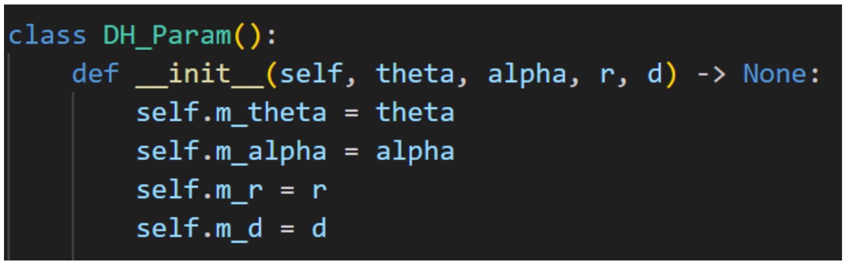
A DH-transformation matrix is defined as:

$$T_{DH}(\theta,\alpha,d,r) = \begin{pmatrix} c\theta & -s\theta * c\theta & s\theta * s\alpha & r * c\theta \\ s\theta & c\theta * c\alpha & -c\theta * s\alpha & r * s\theta \\ 0 & s\alpha & c\alpha & d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Where, $c\theta$ and $s\theta$ is respectively cos (θ) and sin (θ). $c\alpha$ and $s\alpha$ is respecticly cos (α) and $\sin(\alpha)$. Using this definition, the forward kinematics chain can be calculated, defined as:

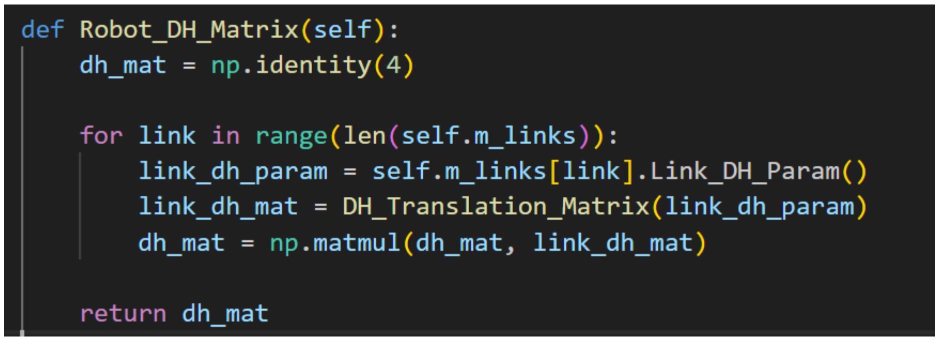
$$FK_{DH} = T_{DH}(\theta_0,\alpha_0,d_0,r_0) * T_{DH}(\theta_1,\alpha_1,d_1,r_1) * T_{DH}(\theta_2,\alpha_2,d_2,r_2) * T_{DH}(\theta_3,\alpha_3,d_3,r_3) * T_{DH}(\theta_3,\alpha_3,d_3,r_3)$$

Where θ_0 denotes the θ value for the link index 0 from Tabell 1. Doing this calculation results in a transformation matrix, where P is the location of the end-effector. The same convention applies for the rest of the parameters. The implantation of this in Python code is shown in Figure 5, Figure 6 and Figure 7. The simulation does the same calculation using C++.



```
class DH_Param():
    def __init__(self, theta, alpha, r, d) -> None:
        self.m_theta = theta
        self.m_alpha = alpha
        self.m_r = r
        self.m_d = d
```

Figure 5 Kin_Math.py DH-parameters class.

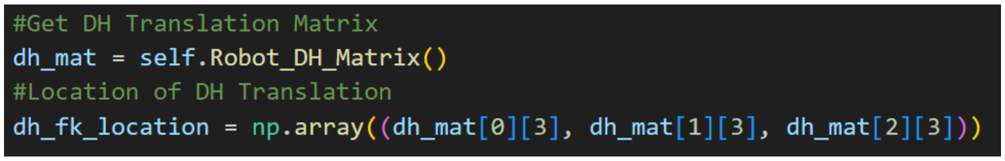


```python
def Robot_DH_Matrix(self):
    dh_mat = np.identity(4)

    for link in range(len(self.m_links)):
        link_dh_param = self.m_links[link].Link_DH_Param()
        link_dh_mat = DH_Translation_Matrix(link_dh_param)
        dh_mat = np.matmul(dh_mat, link_dh_mat)

    return dh_mat
```

Figure 6 Python code for DH forward kinematics.



```python
#Get DH Translation Matrix
dh_mat = self.Robot_DH_Matrix()
#Location of DH Translation
dh_fk_location = np.array((dh_mat[0][3], dh_mat[1][3], dh_mat[2][3]))
```

Figure 7 Location of end-effector from DH forward kinematics.

Simulation

Figure 8 is a screenshot from the simulation running. The simulated robot arm, in orange, is accurately dimensioned to the Tinkerkit robot's dimensions. The user interface includes sliders to adjust the angle (degrees) of each joint in the chain. Moving a slider instantly adjust the robot, so real-time adjust can be made.

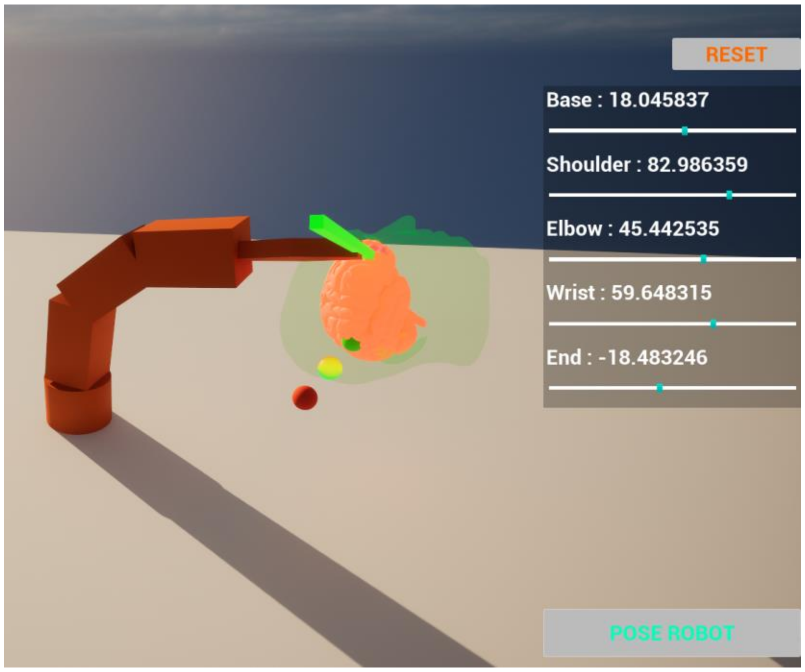
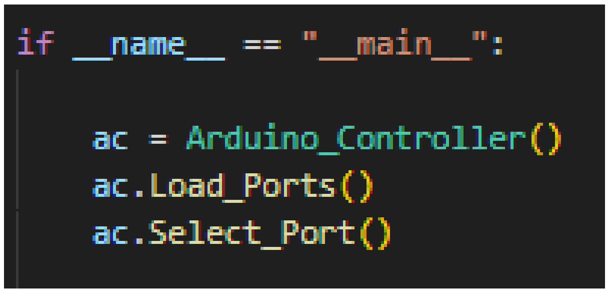


Figure 8 Simulation screengrab

Controlling the Robot

The Python script called "Robot_Controller.py" contains the program which controls the robot arm. The program implements Arduino to Python to UE5-simulation communication, kinematic-calculations and robot-brain spatial information calculations. Kin_math.py is a script referenced by the program, it contains custom written helper-functions for kinematics. See the conclusion section for source code.

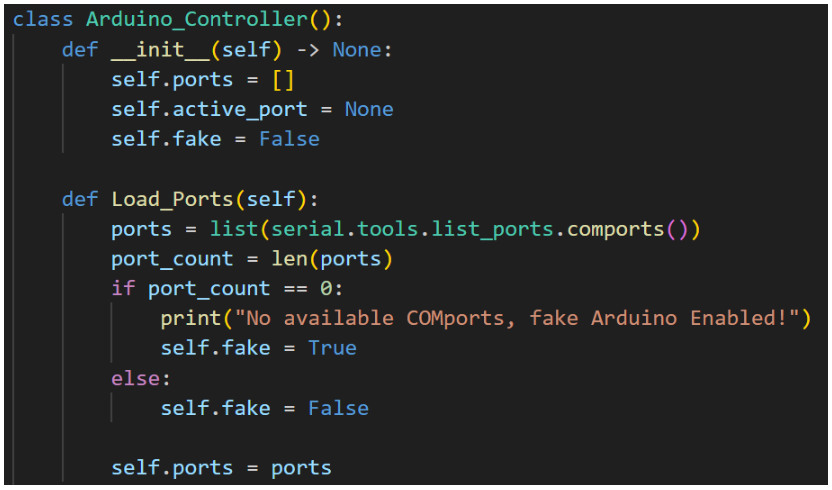
The program implements a class called Ardunio_Controller, one third of it is shown in Figure 10. This class serves as an abstraction-layer for the code specific to the Arduino. The controller loads all the available devices connected to the computer's serial ports. If none are found, the program launches in "fake Arduino"-mode, see Figure 10. This enables the program to be developed, tested and used without a real Arduino connection. If any are found, the user is prompted with the list and chooses what device to connect to.



```
if __name__ == "__main__":

    ac = Arduino_Controller()
    ac.Load_Ports()
    ac.Select_Port()
```

Figure 9 Start of Robot_Controller.py.



```
class Arduino_Controller():
    def __init__(self) -> None:
        self.ports = []
        self.active_port = None
        self.fake = False

    def Load_Ports(self):
        ports = list(serial.tools.list_ports.comports())
        port_count = len(ports)
        if port_count == 0:
            print("No available COMports, fake Arduino Enabled!")
            self.fake = True
        else:
            self.fake = False

        self.ports = ports
```

Figure 10 Arduino Controller class.

Communication

The program communicates with the Arduino via the serial bus, this requires the Arduino to be connected via USB to the computer. Bytes are sent across the serial bus, and therefore the data needs to be encoded and decoded properly for the data to arrive uncorrupted. Therefore, the messages sent from Python to the Arduino are encoded and decoded using the UTF-8 standard.

The simulation utilizes Open Sound Control (OSC), an open-source API with networking features. The simulation hosts an OSC-client during its entire runtime. The client can send messages to an address and port on the local area network. In this case, the server will run on the same computer running Windows 11. Therefore, both the client and server use address 127.0.0.1 (localhost) and port 7800.

The python script is hosting the client’s corresponding OSC-server. The server listens to port 7800. The server continuously monitors the port for new messages. Therefore, the server runs asynchronously from the rest of the python program. This means that the server’s code is run on a separate computing thread, independent from the rest of the program.

Spatial Relations Solution

As stated earlier, real-time tracking of a head’s position and rotation was beyond the scope of this project. Thus, another solution was required. Using the equations defined earlier, we can calculate the location of the target and entry points in relation the robot’s base, or any other frame, but only if the location and rotation of the brain is already known. Therefore, to solve the equations, a method to accurately position the head/brain in relation to the robot’s base was needed.

In the simulation, the brain and robot’s position and rotations are precisely known, meaning that a brain and robot placed with the same relation in the real world, would grant the spatial information needed. This is further covered in the results section.

The Brain

A free to use and distribute 3D model of brain was acquired from Sketchfab. Using Blender, the model was scaled to reflect an average sized brain. A render of the model is shown in Figure 11. A target and entry point are defined. The target is inside the brain, where an electrode should reach. The entry point is where on the scalp it should enter from. A vector for the safe insertion path can then be found as:

$$target - entry$$

Figure 11 Brain 3D-model

Figure 12 shows the brain inside the simulation, it's x-axis is the red arrow, meaning that it's rotated 90 degrees on its y-axis. Therefore, an easy equation is defined to find the target and entry points in robot-space:

$$P_{target} = P_{brain} + R_y(90) * P_{target}$$

Where P_{target} is the location of the target in robot-space, P_{brain} is the location of the brain in robot-space, $R_y(90)$ is a rotation matrix along the y-axis with a rotation of 90 degrees and P_{target} is the location of the target point in brain-space. The same applies for the entry point.

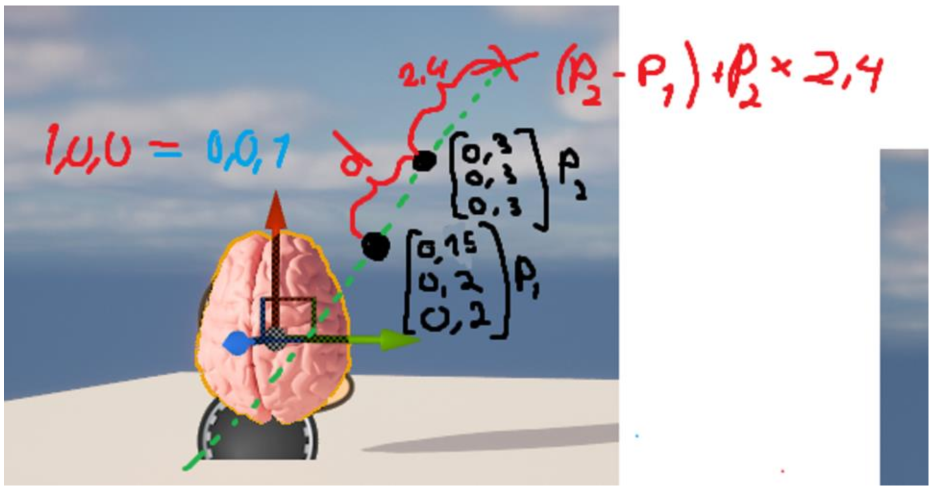


Figure 12 Target and Entry in brain-space

The Head

One of the author’s head was both 3D scanned and printed, as shown in Figure 13. Hair introduces complications for both the scan and print. Therefore, the only bald author’s head was chosen for this task.

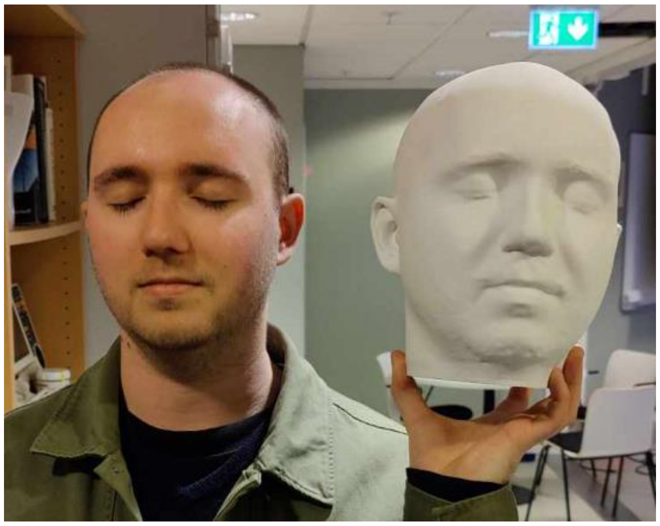


Figure 13 3D printed head.

A stand which is suitable to the head was designed and 3D printed. The stand will determine the position of the head and the brain in the robotic coordinate system. The stand is displayed in Figure 14.

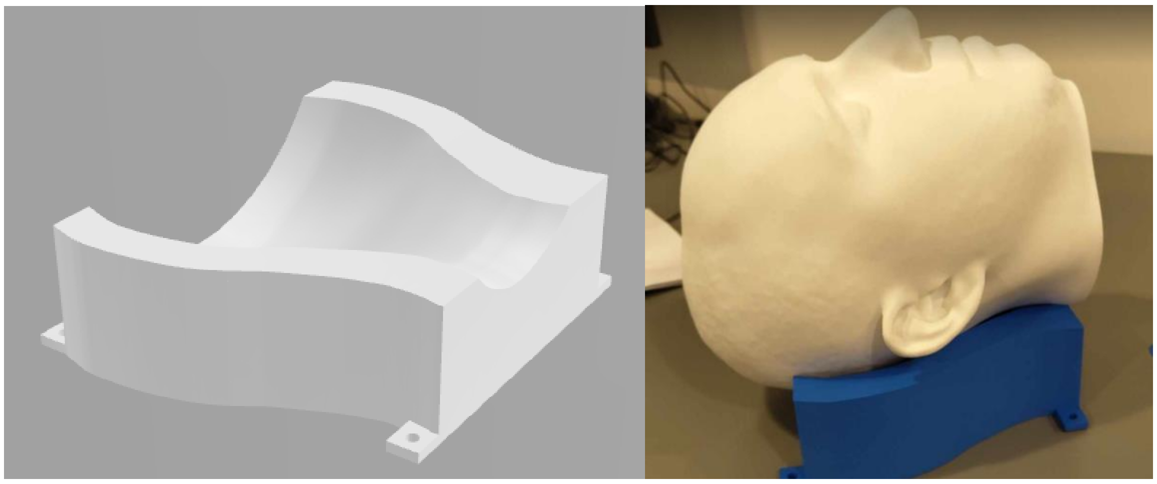


Figure 14 3D printed head on stand

To account for the stand and head, the previous equation is altered slightly:

$$P_{target} = P_{stand} + P_{head} + (R_y(90) * P_{brain}) + P_{target}$$

Where, P_{stand} is the location of the stand in robot-space and P_{head} is the location of the head in stand-space and P_{brain} is now the location of the brain in headspace. The rotation matrix is now applied to the brain instead, due to the head being at 90 degrees in the stand.

Results

The literature review conducted resulted in the conclusion: RADBI systems are safe and accurate, and has great potential for the future (Ma, 2022). The final look of the simulation is shown in Figure 8.

Figure 15 shows the final custom end-effector attached to the robot with the gyroscope attached.

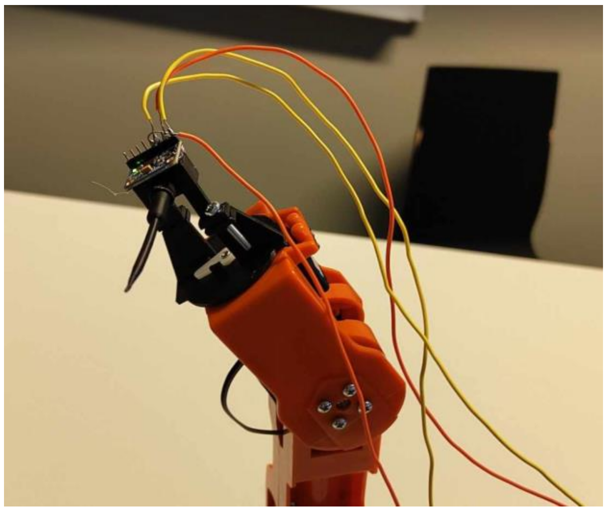


Figure 15 The custom end-effector with gyroscope attached.

Figure 16 shows an overview of the system. The computer is running both the Python program and the simulation. The simulation is missing from the graph, but is would be connected to the computer the same as the Python logo. The connections between Python, simulation, Arduino, gyroscope and robot, are all handled by the Python program. Expect, the user is asked to choose which Arduino to connect to, if more than one is connected to the computer.

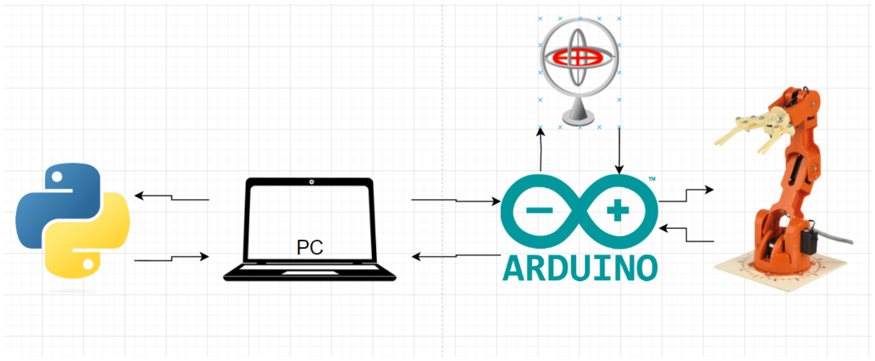


Figure 16 System overview grahpic.

The final process to use the developed system is defined by 7 steps:

1. Connect computer to Arduino via USB.
2. Launch the simulation.
3. Launch Python program.
4. Interface with the program to choose target and entry points.
5. Pose the robot accordingly inside the simulation.
6. Press the “Pose”-button.
7. Wait a few seconds and watch the robot pose itself correctly.

The calculations for inverse kinematics using the DH method proved too difficult and time consuming. However, the calculations done were not wasted. The equations of forward kinematics are used inside the simulation. Without these calculations, accurately positing the robot would be impossible. Another use case is to display an error, the equation is defined:

$$Error = FK_{DH} - P_{ee}$$

Where, FK_{DH} is the location of the end-effector from the DH-model forward kinematics equation, and P_{ee} is the location of the end-effector inside the simulation. This error describes how far away from the target destination the end-effector is on each axis. This error is only possible to calculate due to the kinematic calculations described earlier.

Discussion

Limitations

The largest challenge is how hard it is to operate our equipment in narrow locations. The brain's arteries and nerves must be accurately located, but the Braccio Tinkerkit isn't quite capable of doing this. Consequently, the accuracy of intraoperative visualization and accuracy in approaching the goal, is suboptimal. This is discussed later in the section. Additionally, this project took course over 9 weeks, a short timespan for a complicated system like this.

The simulation and system were developed on and for a powerful computer with a Graphical Processing Unit (GPU). GPUs are not common on microcontroller nor regular commercial laptops. This means that a power computer is required to run the system. The authors do not deem this a major limitation, as per the settings where this kind of system is employed, the authors assume already have access to quite powerful computers with GPUs.

3D Scanning and Printing

3D printing was used for this project. This enabled rapid iterations with great cost-effectiveness. Altogether, the cost of filament is estimated to 400 kr. But, depending on the filament type used, 3D printed parts may not be reliable or strong enough for robotics and industrial applications. Additionally, the design of 3D printed parts is often not possible to manufacture with traditional methods and materials, such as steel prats which takes weeks to manufacture. However, 3D printing parts is entirely suitable for projects like this.

One of the author’s head was 3D scanned. The reliability and accuracy of this method was early on in question, however quickly proved to quite powerful. A few attempts were made, but it only took about 30 minutes to get a satisfactory scan. 3D printed parts are susceptible to shrinking and morphing, but the authors cannot visually notice any divergence from the actual head.

Simulation

In the realm of mathematics, simulations as a tool can open gateways to understanding complex concepts that are otherwise hard to grasp. UE5’s built in features combined with the user’s ability to write custom C++ code and connect it to other programs such as Phytons scripts, is a powerful implantation of this concept. Using the simulation instead of the actual robot, allowed for faster iteration times and testing. Using the actual robot, possibly any other robot, introduces several bottlenecks, such as setup, safety procedures and the laws of nature. In a simulation these constraints can be overcome. However, a simulation made without proper mathematics and methodologies, can moreover impede development and grant false positives to the operators, this could be catastrophic.

Furthermore, UE5’s graphical fidelity is an extremely complicated feature, otherwise almost impossible to realize. Anecdotally, “wow” and “awe” are common first-reactions when not already familiar people get introduced to an UE5 simulation. This may have the advantage of legitimizing concepts, ideas or projects that otherwise would not be paid any attention too.

Accelerometer and Gyroscope

Only the gyroscopic data from the MPU6050 is used. Including data from the accelerometer could open for new techniques for RADBI. For example, instead of using inverse kinematics, a gradient descent function can be applied to the data, to gradually move the robot’s end-effector to the desired location. Furthermore, including more accelerometers and gyroscopes on the robot, could enable several new control methods and data-analysis to make the robot safer. For example, if any accelerometer measures acceleration above a certain threshold, an emergency break is activated.

Ethics

Using robots for surgery has several advantages compared to traditional surgery done completely by humans, such as repeatability, speed and accuracy. However, it also introduces new dangers. During an open-brain surgery, where the brain is exposed, any wrongdoing by the robot, or the operator, can result in catastrophic repercussions for the patient.

No safety-considerations were made for this project. For real-world application of RADBI systems, further research is required to ensure its safety for both patients and operators. Critical safety features may include an emergency-break and safe-zones in the operating room.

The nature of neurotechnology is rapidly changing and evolving. For example, some drugs have been shown to reopen some of the brain's plasticity functions. An adult with these brain functions activated, could gain neuroplasticity levels comparable to children. This example poses several ethical questions, many of which we don't have the answers to. However, neuroplasticity also has the potential to solve long-standing medical problems, such as Alzheimer's disease and epilepsy (Nitsche MA, 2012).

Further Work and Improvements

The most important and powerful method to improve the systems developed, is to ask neurosurgeons what they want, and need. For real world application, this is crucial. This was not done during this project. Engineers and software developers, is unlikely to have the insight and experience to develop RADBI systems alone, emphasizing that RADBI systems are an interdisciplinary and complex field, and must be developed as such. The same kind of system as described in this report can be used for DBS. For example, replacing the end-effector tool with tools that generate electromagnetic fields and can accurately target it to specific brain regions.

As seen in Figure 17, MRI scans can be introduced to the simulation, allowing for more complex visualization. For example, if this example were implemented into the made simulation, the end-effector of the robot can be accurately visualized in real-time, even inside the brain. However, this would require both MRI scans and 3D scan of the patient.

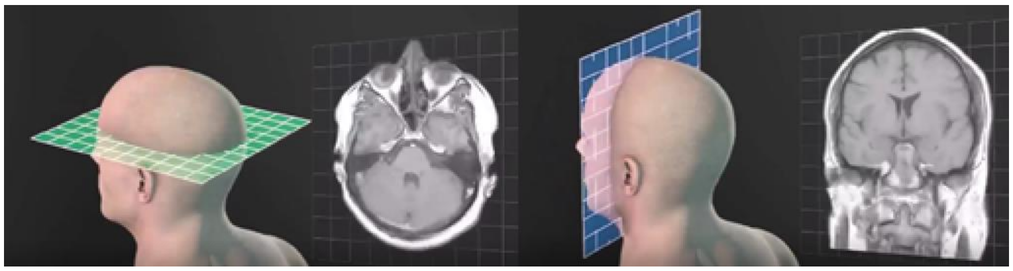


Figure 17Example of MRI scans implemented in a simulation.

Conclusion

Over the course of 9 weeks, the authors have planned, implemented and tested a novel solution to sensory-enabled robot assisted brain implantation. A small-scale literature review was conducted in the preproduction phase. Ethics over RADBI systems and neurotechnology have been discussed. RADBI systems has a promising future and may enable more patients to live happier lives. Several challenges were identified and solved to complete the project. Future improvements and opportunities have been discussed.

The source code the simulation is publicly available at: https://github.com/adamaske/robot_arm_UE_sim

The Python source code is publicly available at:

https://github.com/adamaske/biomedeng/tree/main/Robot_Arm_Project

References

1. Zhe Li. Jian-Guo Zhang, Yan Ye, Xiaoping Li; Review on Factors Affecting Targeting Accuracy of Deep Brain Stimulation Electrode Implantation between 2001 and 2005. Stereotatct Funct Neurosurg 31 January 2017; 94(6): 351-362. https://doi.org/10.1159/000449206
2. Dahl-Hansen, E., Koht, J., & Syvertsen, M. (2019). Epilepsy at different ages-Etiologies in a Norwegian population. *Epilepsia Open*, *4*(1), 176-181. https://doi.org/10.1002/epi4.12292
3. Patel, D. T., Bhanu & Chaunsali, Lata & Sontheimer, Harald. (2019). Neuron–glia interactions in the pathophysiology of epilepsy. *Nature Reviews Neuroscience*, *20*(1). https://doi.org/10.1038/s41583-019-0126-4
4. Nitsche MA, M.-D. F., Paulus W, Ziemann U. (2012). he pharmacology of neuroplasticity induced by non-invasive brain stimulation: building models for the clinical use of CNS active drugs. *J Physiol.*, *590*(19), 4641-4662. https://doi.org/10.1113/jphysiol.2012.232975
5. Ma, F.-Z., Liu, D.-F., Yang, A.-C., Zhang, K., Meng, F.-G., Zhang, J.-G., & Liu, H.-G. (2022). Application of the robot-assisted implantation in deep brain stimulation

[Original Research]. *Frontiers in Neurorobotics, 16*. https://doi.org/10.3389/fnbot.2022.996685