Neuralink Project Presentation: Brain Motion

Problem Statement:
During the surgery process, the surgeon drills a 25 mm diameter circular hole referred to as the "craniectomy." As the lungs inhale and exhale air and as the heart pumps blood in and out of the brain, two kinds of motions that bulges the cortical surface through the craniectomy are produced. The cardiac cycle constitutes shorter pseudo-sinusoidal cycles, whereas the respiratory cycle produces longer pseudo sinusoidal cycles.

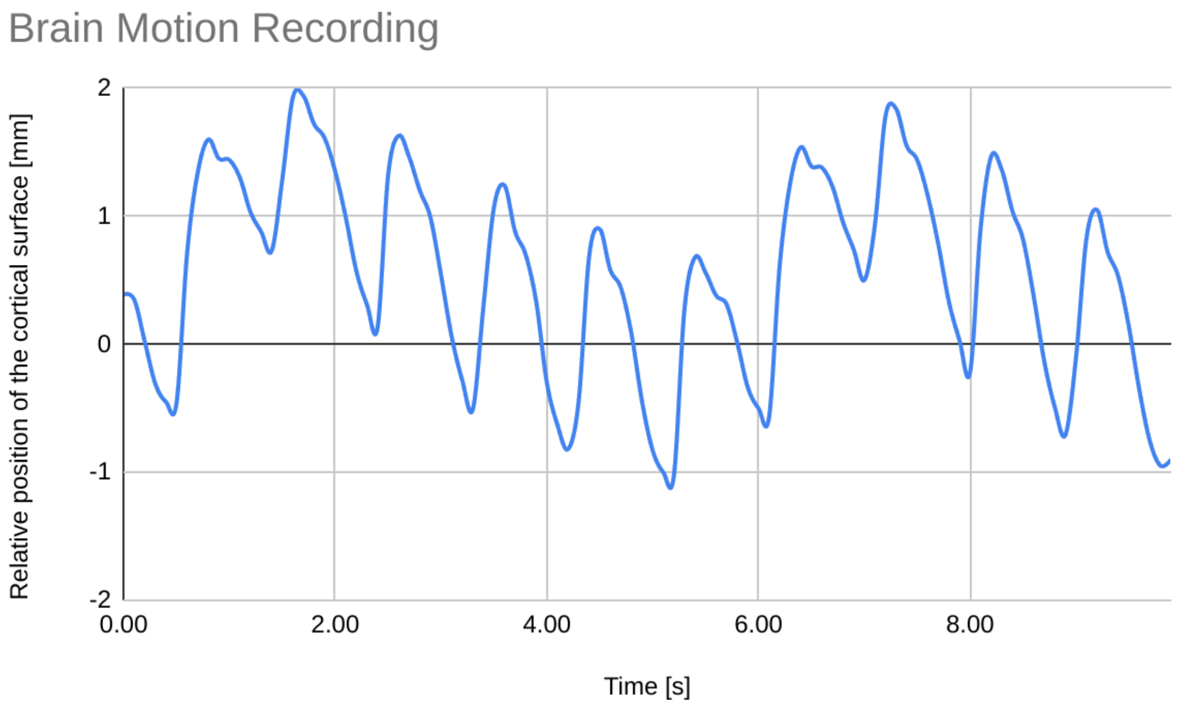

Figure 1. 10 second recording of brain motion observed

The above data is gathered using an Optical Coherence Tomography (OCT) sensor which can return the distance between it and the brain surface. The **Robot Interface** section below provides an interface for this sensor.

The task of the R1 robot system is to insert threads containing electrodes into the cortex while minimizing Z-axis error as much as possible. Z error is defined as the difference between the "commanded depth" (ranging from 3-7mms) and the ground truth depth which is manually determined using in-surgery data.

Assume that the thread is attached to the needle with enough friction force to hold it in place, until the needle retracts from within the cortex, at which point, the friction force between the brain and the thread will leave the thread seated within the brain. Assume that the depth of the thread remains constant after needle retraction.

See the included README.md in the data directory for further information in the data provided.

Robot Specification

A full insertion cycle involves the “pre-insert approach move” which brings the robot close to the cortex. Once it is within some distance (in this example, the distance is 100um but does not have to remain 100um in your solution) from the brain, the needle extends and retracts. The full insertion cycle is portrayed in **Figure 2**.

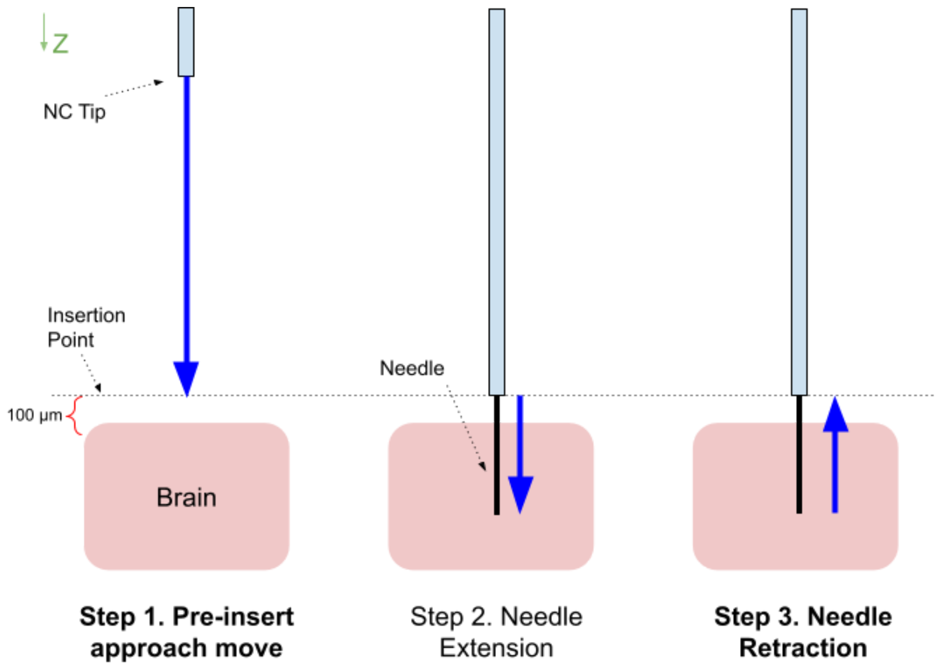

Figure 2. Steps required for a complete “insertion cycle”

Figure 3 is an expanded systems diagram that explains the internal interaction between the surgery control software (which you are tasked with implementing) and the hypothetical robot software (which you ***are not*** tasked with) during steps 2 and 3 of **Figure 2.**

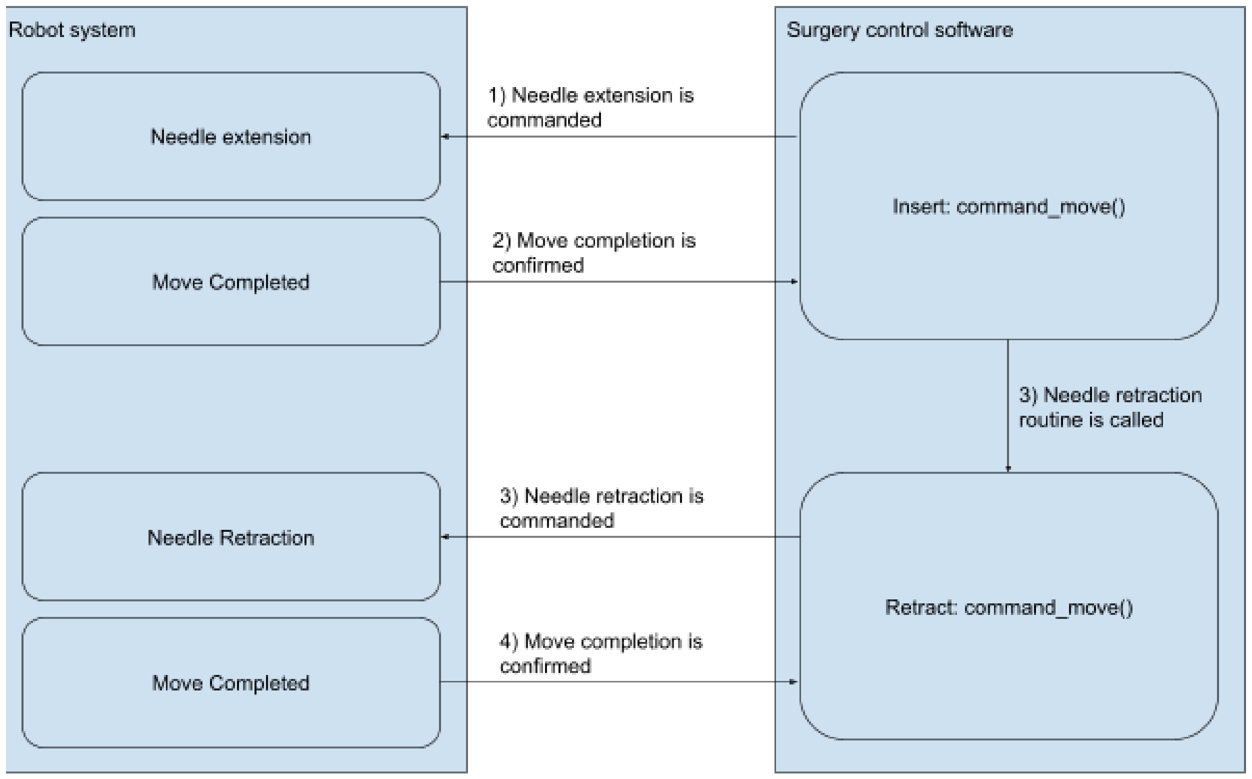
Robot system
Needle extension
Move Completed
Needle Retraction
Move Completed
Surgery control software
Insert: command_move()
Retract: command_move()
1) Needle extension is commanded
2) Move completion is confirmed
3) Needle retraction routine is called
3) Needle retraction is commanded
4) Move completion is confirmed

Figure 3. Expanded systems diagram for Step. 2 and Step 3 of the insertion cycle

Deliverables

1) A design document which is not limited to but must include the following elements:
    a) Background research
    b) Goals and non-goals
    c) Proposed implementation and alternatives
    d) Testing plan
    e) Open questions & future work
2) An implementation of the solution written in Rust.
    a) Since you do not have access to the robot, use the below defined interfaces as a guide to your implementation.
    b) Your implementation should reliably, efficiently, and safely complete steps 1, 2, and 3 in **Figure 3** *at least* 128 times (since there are 128 threads per single N1 implant, but some insertions may face failure modes which should be handled accordingly)
    c) Feel free to create your own implementation of the interfaces defined below for testing purposes
    d) While your implementation of the solution must be written in rust, any supporting work can be done in any language of choice.
3) Change Document which documents your final solution which is not limited to but must include the following elements:
    a) An explanation of your final implementation

b) Evidence for your implementation's safety in a wide range of conditions
c) Evidence for your implementation's efficacy in a wide range of conditions

Your design document and change document can also come in the form of a presentation. Whichever format you prefer. You are expected to give a 45 minute (including 15 minutes Q&A) presentation on your solution, including the contents of the design and change documents.

Please don't hesitate to communicate with the hiring team if there are any further questions about the problem; you are expected to ask questions regarding the question. Please try to batch questions as much as possible to avoid overly frequent emails.

When drafting your deliverables, assume that your software will be used during a human surgery and therefore safety is the utmost priority.

Robot Interface

Unset
```
//! Reference interface definition for the OCT sensor and the simplified robot.
//! You can modify the interfaces and enums provided here, but assume that
other
//! developers will be working with the same interface, meaning any interface
//! changes require sound reasoning and justification.


#[derive(Debug)]
pub enum OCTError {
    // Failed to acquire data from the OCT laser
    AcquisitionError { msg: String },
    // Failed to communicate with the OCT driver
    CommunicationError { msg: String },
    // Timeout waiting for the OCT driver to respond
    TimeoutError { msg: String },
}

/// OCTService provides a high level interface with the OCT sensor.
/// The only function defined here is get_surface_distance which returns
/// the distance between `inserter_z` and the brain surface in nm.
/// the function is async because communication time between the software
/// and the OCT sensor is non-deterministic.
///
/// The initial position of the brain relative to inserter_z is 7mm.
pub trait OCTService {
    // returns the distance between inserter_z and the brain surface in nm
```

```rust
    async fn get_surface_distance(&self) -> Result<u64, OCTError>;
}

pub enum Move {
    InserterZ(u64), // desired absolute position in nm
    NeedleZ(u64),   // desired absolute position in nm
}

/// RobotState represents the current state of the robot where
/// each field represents an axis of our simplified robot.
///  - inserter_z: position of the tip of the needle cartridge which holds the
needle
///  - needle_z: position of the needle tip
///
///  A increase in position indicates movement towards the brain surface
(down),
///  a decrease in position indicates movement away from the brain surface
(up).
pub struct RobotState {
    pub inserter_z: u64, // Absolute encoder position in nm
    pub needle_z: u64,   // Absolute encoder position in nm
}

#[derive(Debug)]
pub enum RobotError {
    // Failed to move the robot
    MoveError { msg: String },
    // lost connection to the robot
    ConnectionError { msg: String },
    // Position exceeds the limits of the robot,
    // can only be thrown by `command_move()`
    PositionError { msg: String },
}

/// Robot provides a high level interface with the robot
/// The simplified robot only has two axes, the tip of the needle cartridge
/// and the needle tip which comes out of the tip of the needle cartridge.
///
/// When a thread is grasped through a successful `command_grasp()` call,
/// the InserterZ axis can be moved in any direction but the NeedleZ axis can
only move
/// in a positive direction.
pub trait Robot {
    async fn get_robot_state(&self) -> Result<RobotState, RobotError>;
    async fn command_move(&self, command: &Move) -> Result<(), RobotError>;
    async fn command_grasp(&self) -> Result<(), RobotError>;
}
```