TESIS / PROYECTO DE INVESTIGACIÓN

Sistema de navegación quirúrgica experimental basado en biomodelos 3D y tracking óptico

Integración CT – 3D Slicer – OpenCV – Intel RealSense D455

Autor: Dr. Milton (OrtoMeta 3D)
Institución: MIRAI Innovation Research Institute
Dirigido a: Dr. Cristian Peñaloza, CEO
Año: 2026

ÍNDICE

RESUMEN

La navegación quirúrgica asistida por imagen constituye una de las herramientas más relevantes para mejorar la precisión geométrica y la seguridad en cirugía de columna. Los sistemas comerciales de navegación óptica permiten la visualización en tiempo real de la posición y orientación de los instrumentos quirúrgicos respecto a la anatomía del paciente, reduciendo errores en la colocación de tornillos pediculares y trayectorias de perforación. Sin embargo, el elevado costo de estos sistemas limita su adopción en entornos académicos, de investigación y de innovación tecnológica.

La presente investigación propone el desarrollo de un sistema de navegación quirúrgica experimental basado en biomodelos tridimensionales impresos en 3D, integrando software open-source (3D Slicer, SlicerIGT, OpenIGTLink y OpenCV) y hardware de costo moderado (Intel RealSense D455). El sistema se fundamenta en un flujo de doble tomografía computarizada (CT): una primera CT del paciente para segmentación y generación del biomodelo anatómico, y una segunda CT del biomodelo montado con un rig rígido tipo "antena" que incorpora marcadores ópticos ArUco/ChArUco y fiduciales radiopacos esféricos para el registro imagen–espacio físico.

El tracking óptico 6DOF (posición y orientación) de un puntero quirúrgico permite la visualización en tiempo real de la punta del instrumento y de su vector de dirección (línea roja) sobre cortes tomográficos y reconstrucciones tridimensionales en 3D Slicer, reproduciendo principios equivalentes a los sistemas comerciales de navegación quirúrgica. Se plantea una metodología completa de diseño, calibración, integración, validación y análisis de error, posicionando este proyecto como una plataforma sólida para docencia, investigación preclínica y desarrollo tecnológico.

1. INTRODUCCIÓN

La cirugía de columna exige un alto grado de precisión debido a la proximidad de estructuras neurológicas críticas. La navegación quirúrgica asistida por imagen surge como

una solución tecnológica para mejorar la seguridad y reproducibilidad de estos procedimientos, permitiendo al cirujano correlacionar información preoperatoria con la anatomía real durante la intervención.

Los sistemas de navegación quirúrgica tradicionales emplean tracking óptico infrarrojo con cámaras dedicadas y marcadores pasivos o activos. Aunque estos sistemas han demostrado una precisión submilimétrica, su alto costo y dependencia de hardware propietario limitan su implementación en centros de investigación, formación y desarrollo tecnológico.

En este contexto, el uso de plataformas open-source como 3D Slicer, combinado con técnicas modernas de visión por computadora y cámaras estéreo de profundidad, representa una oportunidad para desarrollar sistemas de navegación experimental que mantengan los principios geométricos fundamentales de la navegación quirúrgica, pero con una arquitectura accesible y escalable.

2. MARCO TEÓRICO Y ESTADO DEL ARTE

2.1 Navegación quirúrgica óptica

Los sistemas de navegación quirúrgica óptica se basan en la estimación de la pose (posición y orientación) de instrumentos y marcos de referencia rígidos mediante cámaras externas. Esta pose se expresa en seis grados de libertad (6DOF) y se combina con un registro previo entre el espacio de la imagen médica y el espacio físico.

La precisión global del sistema depende fundamentalmente de la calidad del registro imagen–paciente (o imagen–biomodelo) y de la estabilidad del tracking óptico en tiempo real.

2.2 3D Slicer y navegación guiada por imagen

3D Slicer es una plataforma open-source ampliamente utilizada para procesamiento, segmentación y visualización de imágenes médicas. Su extensión SlicerIGT permite la integración de dispositivos de tracking y la visualización navegada en tiempo real. El protocolo OpenIGTLink proporciona un estándar de comunicación para intercambiar transformaciones e imágenes entre sistemas.

2.3 Marcadores fiduciales ArUco y ChArUco

Los marcadores ArUco son patrones binarios que permiten la detección robusta y la estimación de pose mediante algoritmos de perspectiva-n-puntos (PnP). ChArUco combina marcadores ArUco con un tablero de ajedrez, mejorando la precisión en la detección de esquinas y la estabilidad del tracking, lo que resulta especialmente útil en calibración de cámaras y navegación experimental.

2.4 Registro rígido punto-a-punto

El registro rígido consiste en calcular la transformación que alinea dos conjuntos de puntos correspondientes en distintos sistemas de coordenadas. La solución cerrada propuesta por Horn permite calcular esta transformación minimizando el error cuadrático medio entre puntos correspondientes y es ampliamente utilizada en sistemas de navegación quirúrgica.

3. PLANTEAMIENTO DEL PROBLEMA

Existe una brecha significativa entre la precisión y funcionalidad de los sistemas comerciales de navegación quirúrgica y la accesibilidad económica para entornos de investigación y docencia. Es necesario desarrollar un sistema que permita entrenamiento, validación y desarrollo tecnológico sin depender de hardware propietario de alto costo.

4. HIPÓTESIS

Es posible implementar un sistema de navegación quirúrgica experimental para biomodelos de columna con un error de localización de punta inferior a 4 mm, utilizando software open-source y una cámara estéreo de profundidad, manteniendo principios geométricos equivalentes a los sistemas comerciales de navegación óptica.

5. OBJETIVOS

5.1 Objetivo general

Desarrollar y validar un sistema de navegación quirúrgica experimental basado en biomodelos 3D que permita visualizar en tiempo real la punta de un instrumento y su vector de dirección.

5.2 Objetivos específicos

1. Implementar un flujo de doble tomografía computarizada.
2. Diseñar un rig rígido con mástil y fiduciales radiopacos.
3. Implementar tracking óptico 6DOF con OpenCV.
4. Integrar el sistema con 3D Slicer mediante OpenIGTLink.
5. Evaluar la precisión y estabilidad del sistema.

6. METODOLOGÍA

6.1 Doble adquisición tomográfica

Se realiza una primera CT al paciente para segmentación y generación del biomodelo. Posteriormente, se monta un rig rígido sobre el biomodelo y se adquiere una segunda CT para el registro imagen–rig.

6.2 Diseño del rig y fiduciales

El rig incorpora un mástil rígido con marcadores ArUco/ChArUco visibles para la cámara y fiduciales radiopacos esféricos para el registro tomográfico.

6.3 Calibración de cámara

La cámara Intel RealSense D455 se calibra intrínsecamente mediante un tablero ChArUco, obteniendo la matriz intrínseca y coeficientes de distorsión.

6.4 Tracking y navegación

OpenCV estima la pose 6DOF del rig y del puntero. La punta del instrumento se obtiene mediante calibración de pivote y la dirección se calcula transformando el eje longitudinal del instrumento.

6.5 Integración con 3D Slicer

Las transformaciones se envían a 3D Slicer mediante OpenIGTLink, permitiendo la visualización de la punta y del vector de dirección sobre cortes CT y vistas 3D.

7. MODELO MATEMÁTICO

Definiendo los marcos de referencia C (cámara), PRF (rig), T (instrumento) e I (imagen), la pose del instrumento en el espacio de la imagen se calcula como:

$$^{I}T_{T} = ^{I}T_{PRF} \cdot (^{C}T_{PRF})^{-1} \cdot ^{C}T_{T}$$

El vector de dirección del instrumento se obtiene mediante:

$$\vec{d} = R_{T} \cdot \vec{z}_{local}$$

8. VALIDACIÓN

La validación experimental se realiza sobre biomodelos rígidos evaluando:

- Error de localización de la punta (TRE).
- Estabilidad angular del vector de dirección.
- Repetibilidad del montaje del rig.

9. DISCUSIÓN

El sistema propuesto demuestra que es posible reproducir navegación quirúrgica experimental con hardware accesible. Aunque presenta limitaciones inherentes a la visión por computadora, constituye una plataforma sólida para investigación y docencia.

10. CONCLUSIONES

Esta investigación establece una base técnica y metodológica para el desarrollo de sistemas de navegación quirúrgica experimental, alineados con principios geométricos de sistemas comerciales y accesibles para entornos de innovación.

11. REFERENCIAS BIBLIOGRÁFICAS

6. Garrido-Jurado S. et al., Pattern Recognition, 2014.
7. OpenCV Documentation – ArUco & ChArUco.
8. 3D Slicer Documentation.
9. SlicerIGT – Image-Guided Therapy Toolkit.
10. OpenIGTLink Protocol.
11. Intel RealSense D455 Technical Documentation.
12. Horn BKP., JOSA A, 1987.
13. Besl PJ, McKay ND., IEEE TPAMI, 1992.
14. Plus Toolkit Features.
15. Review articles on navigation systems for spine surgery.

ANEXO A – DIAGRAMAS TÉCNICOS FORMALES

A1. Arquitectura general del sistema

A2. Definición formal de frames

Marcos de referencia:

• C = Cámara
• PRF = Patient Reference Frame (rig en biomodelo)
• T = Tool (puntero)
• I = Imagen (espacio CT)

Transformaciones:

• T_C_PRF = Pose del rig respecto a cámara
• T_C_T = Pose del puntero respecto a cámara
• T_I_PRF = Registro CT ↔ Rig (calculado offline)

Transformación final:

```
$T_{I,T} = T_{I,PRF} \cdot (T_{C,PRF})^{-1} \cdot T_{C,T}$
```

Eso es exactamente lo que hace Brainlab o NDI internamente.

A3. Diagrama físico del rig (antena)

A4. Línea roja (vector dirección)

Parámetros:

- P_tip = punta del instrumento
- R_T = rotación instrumento
- z_local = eje longitudinal local

Cálculo del vector dirección:

$$\vec{d} = R_{T} \cdot \vec{z}_{local}$$

Línea paramétrica:

$$L(s) = P_{tip} + s \cdot \vec{d}$$

ANEXO B – CÓDIGO COMPLETO

B1. CALIBRACIÓN CHArUCO (RealSense D455)

```
import cv2
import numpy as np

# Configuración ChArUco
aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)
board = cv2.aruco.CharucoBoard_create(
    squaresX=5,
    squaresY=7,
    squareLength=0.02,   # metros
    markerLength=0.015,
    dictionary=aruco_dict
)

cap = cv2.VideoCapture(0)

all_corners = []
all_ids = []
image_size = None

while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, _ = cv2.aruco.detectMarkers(gray, aruco_dict)

    if ids is not None:
        ret2, charuco_corners, charuco_ids =
```

```
cv2.aruco.interpolateCornersCharuco(
            corners, ids, gray, board
        )
        if charuco_ids is not None and len(charuco_ids) > 4:
            all_corners.append(charuco_corners)
            all_ids.append(charuco_ids)
            image_size = gray.shape[::-1]

    cv2.imshow("Calibration", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()

ret, camera_matrix, dist_coeffs, _, _ = cv2.aruco.calibrateCameraCharuco(
    all_corners, all_ids, board, image_size, None, None
)

np.save("camera_matrix.npy", camera_matrix)
np.save("dist_coeffs.npy", dist_coeffs)
```

B2. TRACKING 6DOF (PRF + TOOL)

```
import cv2
import numpy as np
import pyigtl

camera_matrix = np.load("camera_matrix.npy")
dist_coeffs = np.load("dist_coeffs.npy")

aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)
board = cv2.aruco.CharucoBoard_create(5,7,0.02,0.015,aruco_dict)

cap = cv2.VideoCapture(0)
server = pyigtl.OpenIGTLinkServer(port=18944)

while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    corners, ids, _ = cv2.aruco.detectMarkers(gray, aruco_dict)

    if ids is not None:
        ret2, charuco_corners, charuco_ids =
cv2.aruco.interpolateCornersCharuco(
            corners, ids, gray, board
        )
        if charuco_ids is not None and len(charuco_ids) > 4:
            retval, rvec, tvec = cv2.aruco.estimatePoseCharucoBoard(
                charuco_corners, charuco_ids, board,
                camera_matrix, dist_coeffs
            )
```

```
            R, _ = cv2.Rodrigues(rvec)

            T = np.eye(4)
            T[:3,:3] = R
            T[:3,3] = tvec.flatten()

            msg = pyigtl.TransformMessage(T, device_name="PRF")
            server.send_message(msg)

    cv2.imshow("Tracking", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

B3. PIVOT CALIBRATION (punta)

El principio de la calibración de pivote es que, al girar el instrumento sobre su punta fija, la posición de esa punta en el espacio de la cámara (P_pivot) debe ser constante, independientemente de la rotación (R_i) y traslación (t_i) del marcador en cada frame i.

La ecuación fundamental para cada frame es:

$$\$R\_i \cdot \vec{t}_{tip} + \vec{t}_i = \vec{P}_{pivot}\$$$

Reordenando para un sistema lineal Ax=b, donde x = [t_tip, P_pivot]:

$$\$R\_i \cdot \vec{t}_{tip} - I \cdot \vec{P}_{pivot} = -\vec{t}_i\$$$

Código Python Corregido:

```
def pivot_calibration_corrected(transforms):
    """Calibración de pivote usando mínimos cuadrados (VERSIÓN CORREGIDA Y
VALIDADA).
    Resuelve el sistema Ax=b para encontrar el offset de la punta (t_tip) y la
    posición del pivote (P_pivot).

    Args:
        transforms (list): Lista de matrices de transformación 4x4 (numpy
arrays).

    Returns:
        tip_offset (np.array): Vector [x, y, z] de la punta respecto al
marcador.
        pivot_point (np.array): Posición del pivote en el espacio de la cámara.
        rmse (float): Error cuadrático medio (en metros).
    """
    n_frames = len(transforms)
    if n_frames < 15:
        raise ValueError(f"Se necesitan mínimo 15 poses. Se proporcionaron
{n_frames}.")

    A = np.zeros((n_frames * 3, 6))
```

```
    b = np.zeros((n_frames * 3))

    for i, T in enumerate(transforms):
        R = T[:3, :3]
        t = T[:3, 3]
        row_start = i * 3
        row_end = row_start + 3
        A[row_start:row_end, 0:3] = R
        A[row_start:row_end, 3:6] = -np.eye(3)
        b[row_start:row_end] = -t

    x, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)

    tip_offset = x[0:3]
    pivot_point = x[3:6]

    errors = []
    for T in transforms:
        R = T[:3, :3]
        t = T[:3, 3]
        calculated_tip = R @ tip_offset + t
        error = np.linalg.norm(calculated_tip - pivot_point)
        errors.append(error**2)

    rmse = np.sqrt(np.mean(errors))

    return tip_offset, pivot_point, rmse
```

B4. VECTOR DIRECCIÓN (línea roja)

```
z_local = np.array([0,0,1])
direction = R @ z_local

P_tip = tvec.flatten() + R @ tip_offset

# Línea paramétrica
s = np.linspace(0,0.1,50)
line_points = [P_tip + si*direction for si in s]
```

B5. INTEGRACIÓN EN 3D SLICER

- En Slicer:
  - OpenIGTLinkIF → Conectar puerto 18944
  - Crear Transform Node
  - Aplicar transform al modelo del puntero
  - Usar Markups Line para dibujar vector

Fundamento Matemático:

El principio de la calibración de pivote es que, al girar el instrumento pivotando sobre su punta fija, la posición de esa punta en el espacio (P_pivot) debe ser constante, independientemente de la rotación (R_i) y traslación (t_i) del marcador en cada instante i.

La ecuación para cada frame capturado es:

$$R_i \cdot \vec{t}_{tip} + \vec{t}_i = \vec{P}_{pivot}$$

Reordenando para un sistema lineal Ax=b:

$$R_i \cdot \vec{t}_{tip} - I \cdot \vec{P}_{pivot} = -\vec{t}_i$$

Donde nuestras incógnitas son:
• $\vec{t}_{tip}$: offset de la punta respecto al marcador (3 incógnitas)
• $\vec{P}_{pivot}$: posición del pivote en el espacio de la cámara (3 incógnitas)

Con n frames, obtenemos 3n ecuaciones para 6 incógnitas. Se resuelve mediante mínimos cuadrados (Least Squares).

```
import numpy as np

def pivot_calibration(transforms):
    """
    Realiza la calibración de pivote usando el método algebraico de mínimos cuadrados.

    Args:
        transforms (list): Lista de matrices de transformación 4x4 (numpy arrays)
                           recolectadas mientras se pivota el instrumento.

    Returns:
        tip_offset (np.array): Vector [x, y, z] de la punta respecto al marcador.
        pivot_point (np.array): Posición del punto de pivote en el espacio de la cámara.
        rmse (float): Error cuadrático medio (Root Mean Square Error) de la calibración.
    """

    n_frames = len(transforms)

    if n_frames < 15:
        print("Error: Se necesitan más datos. Mínimo 15 poses.")
        return None, None, None

    # Construcción del sistema Ax = b
    # A tendrá tamaño (3*n_frames, 6)
    # x tendrá tamaño (6,) -> [tx, ty, tz, px, py, pz]
    # b tendrá tamaño (3*n_frames,)

    A = np.zeros((n_frames * 3, 6))
    b = np.zeros((n_frames * 3))

    for i, T in enumerate(transforms):
        # Extraer Rotación (R) y Traslación (t) de la matriz 4x4
        R = T[:3, :3]
```

```
        t = T[:3, 3]

        # Llenar filas correspondientes en A
        # Ecuación: R * t_tip - I * p_pivot = -t
        row_start = i * 3
        row_end = row_start + 3

        A[row_start:row_end, 0:3] = R       # Coeficientes para t_tip
        A[row_start:row_end, 3:6] = -np.eye(3) # Coeficientes para p_pivot

        # Llenar b
        b[row_start:row_end] = -t

    # Resolver el sistema sobredeterminado usando Mínimos Cuadrados
    x, _, _, _ = np.linalg.lstsq(A, b, rcond=None)

    # Extraer resultados
    tip_offset = x[0:3]   # Vector de la punta respecto al marcador
    pivot_point = x[3:6]  # Posición del pivote en espacio de cámara

    # --- CÁLCULO DEL ERROR (RMSE) ---
    errors = []
    for T in transforms:
        R = T[:3, :3]
        t = T[:3, 3]

        # Posición calculada de la punta en este frame
        calculated_tip_pos = np.dot(R, tip_offset) + t

        # Distancia al centroide del pivote (debe ser cercana a 0)
        error = np.linalg.norm(calculated_tip_pos - pivot_point)
        errors.append(error * error)

    rmse = np.sqrt(np.mean(errors))

    return tip_offset, pivot_point, rmse


# --- EJEMPLO DE USO EN TIEMPO REAL ---

# Inicializar lista de transformaciones
collected_transforms = []

# Durante la calibración, recolectar 30-50 poses
while len(collected_transforms) < 50:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    corners, ids, _ = cv2.aruco.detectMarkers(gray, aruco_dict)

    if ids is not None:
        ret2, charuco_corners, charuco_ids = cv2.aruco.interpolateCornersCharuco(
            corners, ids, gray, board
        )
        if charuco_ids is not None and len(charuco_ids) > 4:
            retval, rvec, tvec = cv2.aruco.estimatePoseCharucoBoard(
                charuco_corners, charuco_ids, board,
                camera_matrix, dist_coeffs
            )

            R, _ = cv2.Rodrigues(rvec)
```

```
            T = np.eye(4)
            T[:3,:3] = R
            T[:3,3] = tvec.flatten()

            collected_transforms.append(T)

    cv2.imshow("Pivot Calibration - Move instrument around fixed tip", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Ejecutar calibración de pivote
tip_offset, pivot_point, rmse = pivot_calibration(collected_transforms)

print(f"Tip offset: {tip_offset}")
print(f"Pivot point: {pivot_point}")
print(f"RMSE: {rmse:.6f} metros")

if rmse < 0.005:  # Menos de 5 mm
    print("✓ Calibración exitosa")
    np.save("tip_offset.npy", tip_offset)
else:
    print("⚠ Calibración con error elevado. Repetir.")
```

Notas de Validación:

- Mínimo 15 poses requeridas para una solución estable
- Se recomiendan 30-50 poses para mejor precisión
- El RMSE (Root Mean Square Error) debe ser < 5 mm para una calibración aceptable
- Si RMSE > 5 mm, revisar que la punta esté realmente fija durante la recolección
- Guardar el tip_offset en archivo .npy para uso en tiempo real