Integración de base móvil a robot baxter para teleoperación en Realidad virtual

C.A. Veintimilla-Sánchez, P.P. Carrasco-González
cveintimillas@gmail.com, pedroprimo2010@hotmail.com
Departamento de Mecatrónica, Tecnológico de Monterrey
Junio 9, 2023, Monterrey, México

Resumen
Este trabajo presenta la integración de una base móvil a un robot colaborativo Baxter para teleoperación mediante lentes y controles de realidad virtual transmitidos a través de internet. La conexión e interpretación de movimientos vienen dados por la posición final de los controladores de los Oculus Quest en el espacio y la transmisión del sistema de visión de la cámara del robot a través de una red privada conectada a un almacenador de datos en la nube a tiempo real resolviendo los movimientos del robot de forma automática con cinemática inversa implementada en paquetes de ROS.

KeyWords: Robótica Teleoperada, Baxter Robot, Realidad Virtual, ROS

Introducción

La robótica teleoperada ha cobrado relevancia en los últimos años con los avances de la robótica y la Inteligencia Artificial. La robótica teleoperada puede aportar en el desarrollo de diferentes áreas, desde la mejora en las habilidades/destrezas de robots avanzados con ayuda del machine learning hasta realizar tareas donde se pueda ver comprometida la salud del ser humano. Creemos que la robótica teleoperada puede ser un pilar fundamental dentro de la exploración espacial y de la manufactura en microgravedad, donde se puedan realizar experimentos sin la presencia humana a cualquier hora y en cualquier momento. Para ello se realizó la conexión y teleoperación de un robot Baxter a través de una red de internet y unos Oculus Quest VR, más una integración de una base móvil para que el robot pueda desplazarse en superficies planas a una velocidad máxima de 40 km/h. Para el control del robot se censaron los datos obtenidos de los sensores inerciales que llevan los controladores del Oculus y se enviaron a un almacenador de datos a tiempo real en la nube, Firebase, que el robot lee como posición final de los grippers resolviendo su cinemática inversa de forma automática replicando la pose final del operario. De la misma forma dicha transmisión también obtiene datos de la posición de la cabeza del operario y la transmisión a tiempo real del sistema de visión del Baxter para que el operario pueda ver en todo momento lo que está viendo el robot. Este proyecto se enfoca en las bases para la integración de un robot completamente teleoperado para futuros avances dentro la estabilidad de la conexión, reducción de latencias y de retrasos en los movimientos.

Marco teórico

La robótica teleoperada se define como el control remoto por el ser humano. Las señales de control remoto pueden ser enviadas a través de cables o a través de un sistema local sin cable como por WiFi sobre internet o por toma satelital. [1]

Dentro de nuestro proceso de integración se consideraron diferentes aspectos en cada implementación realizada. Consistiendo en un robot de brazos colaborativos Baxter de la marca Rethink Robotics [2] y una base móvil reacondicionada de una silla eléctrica. El diseño inicial del sistema fue descrito en [3] y la integración de un robot industrial sobre una silla eléctrica [4] para aplicaciones industriales. Nuestro enfoque era de forma básica la interacción del humano robot de forma teleoperada, para futuras mejoras en el ámbito, a partir de la recopilación de datos de los sensores de los controladores de los oculus quest enviados a una base de datos a tiempo real para el control de los brazos colaborativos del robot Baxter y su cámara. Para ello se evaluaron las características principales como grados de libertad, cantidad de sensores, payload, peso y dimensiones. La habilitación de una base móvil en la que el robot pueda desplazarse sobre superficies planas controlado a través de los controladores del oculus.

Robot colaborativo

El Robot Baxter cuenta con 2 brazos colaborativos con 7 grados de libertad cada uno y un sistema de visión integrada con un peso aproximado de 75 kg sin considerar el pedestal sobre el que está expuesto y un payload por brazo de alrededor de 5 lb. [5]

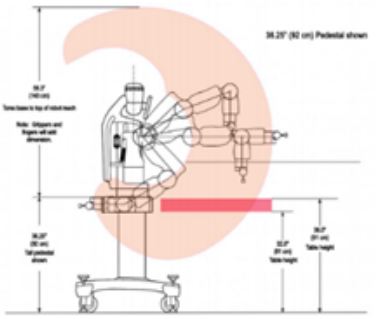


Figura 1a. Área de trabajo de los brazos vista perfil

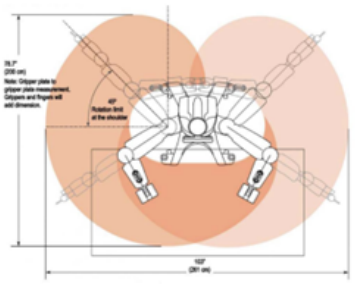


Figura 1b. Área de trabajo de los brazos vista superior

Para realizar los movimientos de los brazos se resolvieron ecuaciones de cinemática inversa para obtener la posición final de los grippers según los datos leídos del Firebase.

La cinemática inversa dada la pose (posición y orientación) del marco de interés, se calculan los valores de las articulaciones para obtener la pose. [6]

Ecuación (1)

$$f_1(\Theta) = L_1c_1 + L_2c_1c_2 - L_3(s_1s_3 + c_1s_2c_3) - L_4((s_1s_3 + c_1s_2c_3)s_4 - c_1c_2c_4) - L_5((s_1c_3 - c_1s_2s_3)s_5 + ((s_1s_3 + c_1s_2c_3)c_4 + c_1c_2s_4)c_5) - x_7^0$$

Ecuación (2)

$$f_2(\Theta) = L_1s_1 + L_2s_1c_2 + L_3(c_1s_3 - s_1s_2c_3) + L_4((c_1s_3 - s_1s_2c_3)s_4 - s_1c_2c_4) + L_5((c_1c_3 + s_1s_2s_3)s_5 + ((c_1s_3 - s_1s_2c_3)c_4 - s_1c_2s_4)c_5) - y_7^0$$

Ecuación (3)

$$f_3(\Theta) = - L_2s_2 - L_3c_2c_3 - L_4(s_2c_4 + c_2c_3s_4)$$

$+ L_5((s_2s_4 - c_2c_3c_4)c_5 + c_2s_3s_5) - z_7^0$

Matrices de Rotación

Ecuación (4)
$f_4(\Theta) = -((s_2s_3 + c_1s_2c_3)s_4 - c_1c_2c_4)c_6 - ((s_1c_3 - c_1s_2s$
$+ ((s_1s_3 + c_1s_2c_3)c_4 + c_1c_2s_4)c_5)s_6 - r_{13}$

Ecuación (5)
$f_5(\Theta) = ((c_1s_3 + s_1s_2c_3)s_4 + s_1c_2c_4)c_6 + ((c_1c_3 - s_1s_2s_3)$
$s_5 + ((c_1s_3 - s_1s_2c_3)c_4 - s_1c_2s_4)c_5)s_6 - r_{23}$

Ecuación (6)
$f_6(\Theta) = -((s_2s_4 - c_2c_3c_4)s_5 - c_2s_3c_5)c_7 - ((s_2c_4 - c_2c_3s$
$+ ((s_2s_4 + c_2c_3c_4)c_5 + c_2s_3s_5)c_6)s_7 - r_{32}$

Base Móvil

Para la base móvil se utilizó una silla eléctrica comercial con un soporte máximo de 100 kg. Dentro de la base móvil se consideraron diferentes análisis de estructura como flexión y peso en superficies planas para garantizar la estabilidad y soporte del robot dentro de la plataforma a realizar para añadir a la silla eléctrica y fijar el robot Baxter, tomando en cuenta el espacio de trabajo de los brazos y su carga máxima.

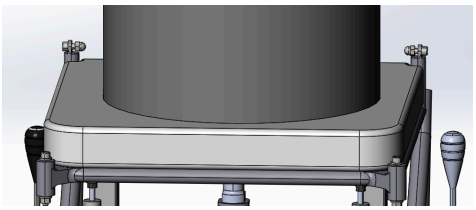


Figura 2. Cálculo de fuerzas Normal y peso sobre superficies planas

$\overline{P} = m \cdot - g$ *Ecuación (7)*
$\overline{N} = - \overline{P}$ *Ecuación (8)*

Figura 3. Módulo de flexión

Tornillo de Aluminio aproximadamente de 20 mm y 4 mm de saliente

$A = \pi \cdot r^2$ *Ecuación (9)*
$S = \frac{\sigma}{Tan\phi} = \frac{\frac{F}{A}}{\frac{d}{l}}$ *Ecuación (10)*

Control del operario

Para los movimientos de los brazos y cabeza de forma teleoperada se utilizaron con los oculus Quest que utilizan un sistema de posición llamado Inside-out Tracking [8] que usa cámaras y aprendizaje artificial para llevar a cabo la medición y el seguimiento preciso de la posición de los controles. Por otro lado, los controladores cuentan con un sistema de led infrarrojos en sus anillos y dentro del control un sistema de sensores inerciales, al igual que en su visor. Las cámaras del visor detectan dichos leds y continuamente toman imágenes de ellos para tener su posición supervisada, basadas en estas imágenes la llamada Constellation tracking system triangula la posición de los controles en el espacio. [8]

Los Oculus Quest tienen su entorno de desarrollo y pueden ser conectados mediante API a Unity que es un motor gráfico de videojuegos que facilitó el desarrollo del seguimiento de cámara para transmitir lo que está viendo el robot. [9]

Figura 4. Sensores Infrarrojos Oculus Quest

Almacenamiento y actualización de Datos

Los datos obtenidos de los sensores de los controladores del oculus quest son enviados a un Firebase mediante WiFi. Firebase es una plataforma de desarrollo que cuenta con diferentes productos de base de datos, servicios, autenticación en la nube [10]. Específicamente para este proyecto se utilizó el almacenamiento y sincronización de datos en tiempo real de los sensores de los oculus y sus controladores para que el robot conectado a la misma red de internet pueda leer estos datos a tiempo real y redirigir la posición de la cabeza y sus gripes a la posición final de la posición del operador.

Desarrollo

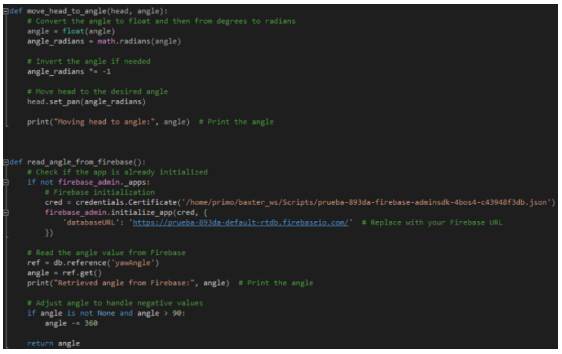
Esta sección del documento presenta la integración de los elementos mecánicos, de control y software que resultan en la teleoperación del robot con un casco de realidad virtual. Se muestra el esquema de control, que a su vez muestra la comunicación entre elementos y el flujo de datos. La explicación de los scripts de código desarrollados también se presentan en esta sección, así como los experimentos realizados.

A. Comunicación con Robot Baxter

Debido a la accesibilidad y amplia documentación del robot Baxter, comunicarse con él es sencillo. El equipo siguió las instrucciones disponibles en la wiki de Rethink Robotics que describe cómo instalar el software necesario y comunicarse con el robot [11]. Para esto, se utilizó una laptop con Ubuntu Xenial y ROS Kinetic. Los scripts que desarrollo el equipo para interactuar con el robot se muestran en la Fig. 1 y Fig. 2 que pertenecen al movimiento de la cabeza y las manos, respectivamente. Son scripts escritos en Python y estos inicializan la conexión con la base de datos y obtienen el dato que se necesita según cada aplicación. El script de la cabeza contiene una posición inicial de los brazos para asegurar que siempre empiece a moverse desde el mismo punto. Los scripts de los brazos toman la posición actual del robot y la definen como un cero local, pues los datos que envía el usuario a la base corresponden a desplazamientos en los ejes de posición y rotación. El script después aplica estos desplazamientos que lee de la base a la posición original del robot para resolver la cinemática inversa y llevar el efector del robot hacia la posición deseada, según el desplazamiento registrado.

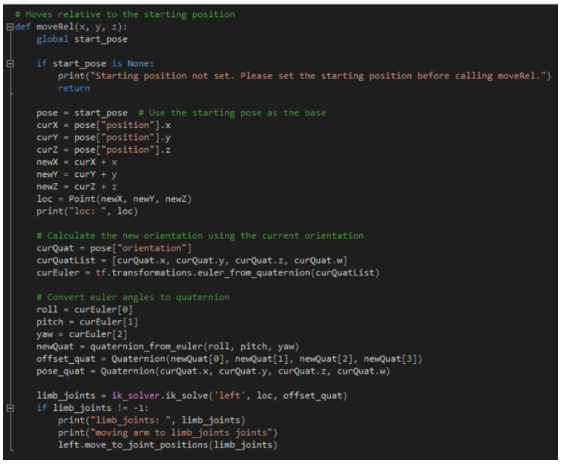
B. Comunicación Meta Quest 2 y Unity

La instalación del software propietario de Meta "Oculus Software" es necesaria para que se pueda conectar el casco de realidad virtual con una workstation. En el editor de Unity, se necesita descargar el paquete de Oculus para habilitar la comunicación entre el editor y el casco. El pedazo de código que se encarga de pedir los datos al casco de realidad virtual se observa en la Figura 3.



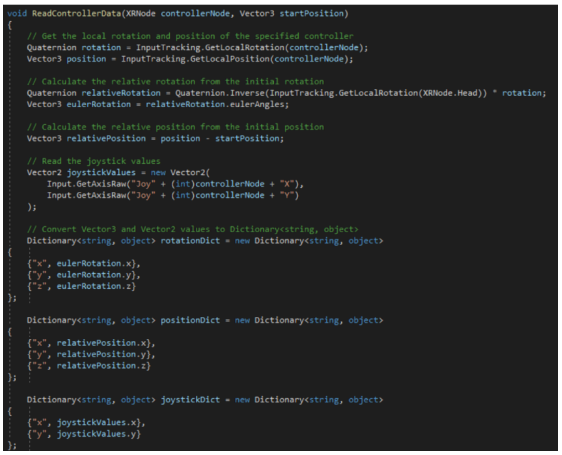
```
def move_head_to_angle(head, angle):
    # Convert the angle to float and then from degrees to radians
    angle = float(angle)
    angle_radians = math.radians(angle)

    # Invert the angle if needed
    angle_radians *= -1

    # Move head to the desired angle
    head.set_pan(angle_radians)

    print("Moving head to angle:", angle)  # Print the angle


def read_angle_from_firebase():
    # Check if the app is already initialized
    if not firebase_admin._apps:
        # Firebase initialization
        cred = credentials.Certificate('/home/primo/baxter_ws/Scripts/prueba-893da-firebase-adminsdk-4bos4-c43948f3db.json')
        firebase_admin.initialize_app(cred, {
            'databaseURL': 'https://prueba-893da-default-rtdb.firebaseio.com/'  # Replace with your Firebase URL
        })

    # Read the angle value from Firebase
    ref = db.reference('yawAngle')
    angle = ref.get()
    print("Retrieved angle from Firebase:", angle)  # Print the angle

    # Adjust angle to handle negative values
    if angle is not None and angle > 90:
        angle -= 360

    return angle
```

Figura 5. Función de lectura de datos de Firebase y mueve la cabeza del robot del ángulo correspondiente



```
# Moves relative to the starting position
def moveRel(x, y, z):
    global start_pose

    if start_pose is None:
        print("Starting position not set. Please set the starting position before calling moveRel.")
        return

    pose = start_pose  # Use the starting pose as the base
    curX = pose["position"].x
    curY = pose["position"].y
    curZ = pose["position"].z
    newX = curX + x
    newY = curY + y
    newZ = curZ + z
    loc = Point(newX, newY, newZ)
    print("loc: ", loc)

    # Calculate the new orientation using the current orientation
    curQuat = pose["orientation"]
    curQuatList = [curQuat.x, curQuat.y, curQuat.z, curQuat.w]
    curEuler = tf.transformations.euler_from_quaternion(curQuatList)

    # Convert euler angles to quaternion
    roll = curEuler[0]
    pitch = curEuler[1]
    yaw = curEuler[2]
    newQuat = quaternion_from_euler(roll, pitch, yaw)
    offset_quat = Quaternion(newQuat[0], newQuat[1], newQuat[2], newQuat[3])
    pose_quat = Quaternion(curQuat.x, curQuat.y, curQuat.z, curQuat.w)

    limb_joints = ik_solver.ik_solve('left', loc, offset_quat)
    if limb_joints != -1:
        print("limb_joints: ", limb_joints)
        print("moving arm to limb_joints joints")
        left.move_to_joint_positions(limb_joints)
```

Figura 6. Función que controla las manos del robot



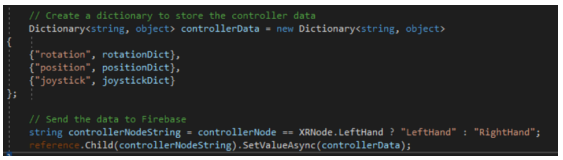
```
void ReadControllerData(XRNode controllerNode, Vector3 startPosition)
{
    // Get the local rotation and position of the specified controller
    Quaternion rotation = InputTracking.GetLocalRotation(controllerNode);
    Vector3 position = InputTracking.GetLocalPosition(controllerNode);

    // Calculate the relative rotation from the initial rotation
    Quaternion relativeRotation = Quaternion.Inverse(InputTracking.GetLocalRotation(XRNode.Head)) * rotation;
    Vector3 eulerRotation = relativeRotation.eulerAngles;

    // Calculate the relative position from the initial position
    Vector3 relativePosition = position - startPosition;

    // Read the joystick values
    Vector2 joystickValues = new Vector2(
        Input.GetAxisRaw("Joy" + (int)controllerNode + "X"),
        Input.GetAxisRaw("Joy" + (int)controllerNode + "Y")
    );

    // Convert Vector3 and Vector2 values to Dictionary<string, object>
    Dictionary<string, object> rotationDict = new Dictionary<string, object>
    {
        {"x", eulerRotation.x},
        {"y", eulerRotation.y},
        {"z", eulerRotation.z}
    };

    Dictionary<string, object> positionDict = new Dictionary<string, object>
    {
        {"x", relativePosition.x},
        {"y", relativePosition.y},
        {"z", relativePosition.z}
    };

    Dictionary<string, object> joystickDict = new Dictionary<string, object>
    {
        {"x", joystickValues.x},
        {"y", joystickValues.y}
    };
```

Figura 7. Función que obtiene los datos de los sensores de Meta Quest 2

Es importante mencionar que el equipo dice enviar los datos como desplazamientos, en el mismo caso que en el código del robot, se declara la posición inicial de los controles de realidad virtual como ceros locales y lo que se envía a la base de datos es el desplazamiento en metros, tanto de posición como de rotación, de esa posición inicial.

C. Comunicación Unity y Firebase

Una vez que el equipó creó una base de datos de tiempo real en Google FireBase, se debe añadir la aplicación de Unity a la base de datos. El equipo siguió la documentación adecuada para dar de alta la aplicación de Unity y otorgarle permisos de escritura y lectura en FireBase [10]. En la Fig. 4 se muestran las instrucciones necesarias para enviar datos a la base una vez que ya se ha registrado la aplicación de Unity en la la consola de FireBase.



```
// Create a dictionary to store the controller data
Dictionary<string, object> controllerData = new Dictionary<string, object>
{
    {"rotation", rotationDict},
    {"position", positionDict},
    {"joystick", joystickDict}
};

// Send the data to Firebase
string controllerNodeString = controllerNode == XRNode.LeftHand ? "LeftHand" : "RightHand";
reference.Child(controllerNodeString).SetValueAsync(controllerData);
```

Figura 8. Envío de datos previamente registrados a la base de datos.

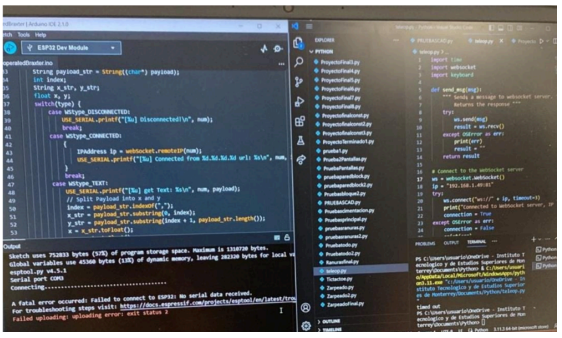


Figura 9. Conexión WebSocket Esp32

D. Comunicación computadora ROS y Workstation con Unity

La computadora que tiene ROS y controla al robot debe tener comunicación de un solo sentido con la workstation de Unity. Esto se debe a que la computadora con ROS es la que tiene acceso a las cámaras del robot, pues es la que está conectada directamente a él. Gracias a los

paquetes de ROS, se abre un servidor local en la ip de la computadora con ROS, este se trata de un web server que almacena la imagen de vídeo de las cámaras del robot. En la workstation con Unity, se accede al servidor creado por la computadora con ROS y se obtiene la liga al feed de vídeo de las cámaras del robot, que después se muestran en el ambiente de realidad virtual.

E. Base Móvil

La base móvil es una silla eléctrica comercial con dos motores en la parte posterior que operan a 24 voltios y 260 vatios y dos ruedas unidireccionales controladas por un joystick como se ve en el proyecto [7]. Dentro del control del joystick se encuentra el circuito integrado que genera las señales analógicas para el control de los motores. Para reemplazar el joystick que se opera de forma manual se implementó un esp32 para mandar la señal vía websocket al controlador y poder mover la silla de forma remota con el script que se muestra en la Fig. 5.

La silla eléctrica cuenta con una autonomía en su batería de 6 a 8 horas dependiendo del uso y la velocidad y aguanta un peso máximo de 100 kilogramos. Para verificar la estabilidad y el soporte de la estructura implementada se calculó el peso y fuerza sobre la estructura y el estrés cortante que siente la barra que detiene el marco de la silla. El conjunto de ecuación (1) muestra el cálculo del peso y la fuerza, donde P corresponde al peso, m la masa, g la fuerza de gravedad y la N la fuerza normal sobre la base del robot debido a su peso.

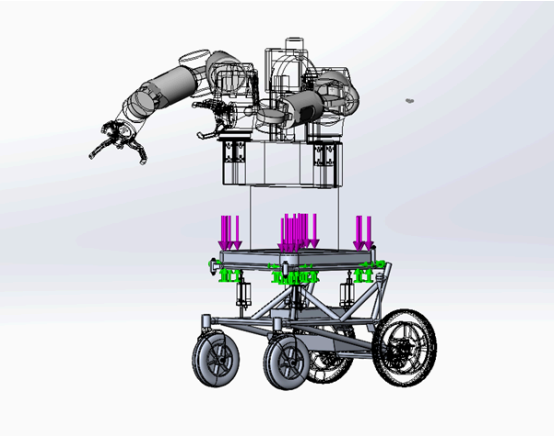


Figura 10. Simulación de esfuerzos cortante que siente la base del robot debido a su peso

1. $\overline{P} = m \cdot - g \qquad \overline{N} = - \overline{P}$

$\overline{P} = 74\,kg \cdot - 9.81 \frac{m}{s^2} \qquad \overline{N} = - (- 725.94N)$

$\overline{P} = - 725.94\,N \qquad \overline{N} = 725.94\,N$

Para el cálculo de la deformación d que sufre la barra de aluminio que mantiene fijo el robot a la base se considera que tiene un radio r de 10 milímetros y una longitud l de 4 milímetros. Se conoce que el aluminio tiene un módulo de corte S de 23,700 megapascales. Se calcula el área A en (2) y después se despeja d en la ecuación del módulo de corte como se muestra en (3) para determinar que la barra se desplaza 3.9×10^{-6} metros.

2. $A = \pi \cdot r^2$

$A = 3.1415 \times 10^{-4} m^2$

$S_{Al} = 23700\,MPa$

3. $S = \frac{\sigma}{Tan\phi} = \frac{\frac{F}{A}}{\frac{d}{l}} \qquad S = \frac{Fl}{Ad}$

$$d = \frac{(725.94\,N)(0.04m)}{(3.1415\times10^{-4}m^2)(23700\times10^{-6}Pa)}$$

Después se realizó un análisis estructural en SolidWorks para validar la estabilidad del robot en la base móvil. Las fuerzas resultantes del análisis estructural se observan en la Fig. 10

F. Sistema de Control

Previamente se mencionó que todas las comunicaciones, a excepción del envío de vídeo de la laptop con ROS a la workstation de Unity, deben ser retroalimentadas. Esto se debe a que todas esas señales causan un cambio en el sistema, a las cuales el elemento que origina la señal debe reaccionar y evaluar. Debido a que se trata de una tarea de teleoperación robótica, no todos los elementos se encuentran conectados a la misma red, sino que se transmiten datos por Internet. El flujo de datos, las comunicaciones entre elementos y las redes a las que pertenece cada elemento se muestran en la Fig. 11

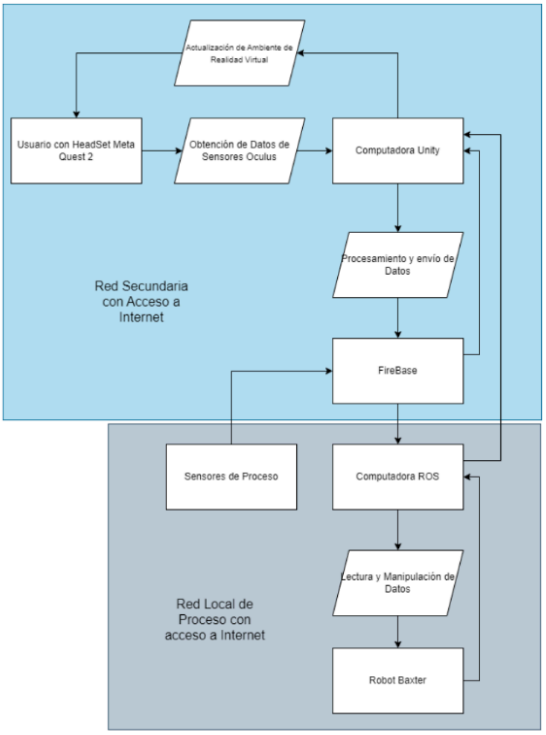


Figura 11. Diagrama de Control del Robot Teleoperado

G. Diseño de Experimentos

Debido a que el editor de Unity y el ambiente de programación de ROS del robot cuentan con sistemas de referencia de coordenadas diferentes, se realizó un experimento para sincronizarlos. En primera instancia, el equipo realizó una prueba con la aplicación de Unity, en donde se activaba el registro de datos de posición y orientación de los sensores. Se observó el cambio en las coordenadas de la base de datos al mover los controles en todos sus grados de libertad para determinar que el eje z es el frente, x la derecha y y hacia arriba en el editor de Unity, además de sus respectivos sentidos positivos de rotación.

De igual manera, una vez que el script que mueve el robot según lo que lee en la base de datos estaba listo, se hizo la prueba y se registró el movimiento del robot al ingresar datos manualmente en cada uno de los ejes de posición y rotación de la base. Así se determinó que el sistema de referencia de coordenadas del robot es x al frente, y a la izquierda y z hacia arriba, así como sus respectivos sentidos positivos de rotación. Con esta información el equipo ajustó en el código la asignación de coordenadas leídas de la base con la coordenada enviada a la función de movimiento para asegurar que las posiciones y rotaciones de movimiento del humano se reflejen correctamente en el movimiento del robot.

Resultados

Dentro de la integración de todos los componentes se pudo realizar la conexión estable de los controladores y el visor a la base de datos en Firebase para que el robot realice lectura de los datos y se posiciones como efecto final en el espacio según el movimiento del operador. El tiempo de respuesta podría variar según la estabilidad de la red, al usar una red de proveedor, en ciertas ocasiones tenía interferencias y baja

transmisión de datos, lo que causaba un retraso en el tiempo de respuesta del robot según los movimientos del operario. Por otro lado, la base móvil fue integrada satisfactoriamente; sin embargo, hubo problemas con la acción del sistema. La conexión de la Esp32 mediante websocket funcionaba correctamente, pero la respuesta del controlador de la silla tenía ciertas complicaciones, es un área que se debe corregir para mejorar la respuesta general del sistema. En general, el prototipo funciona correctamente a pesar de las áreas de mejora, cumple la función requerida de forma correcta.

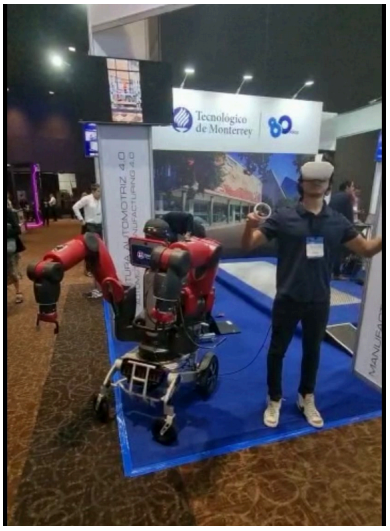


Figura 12. Pruebas de movimiento con los controles del Oculus

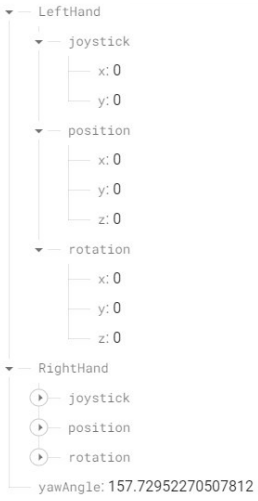


Figura 13. Subdivisión y coordenadas de las diferentes articulaciones

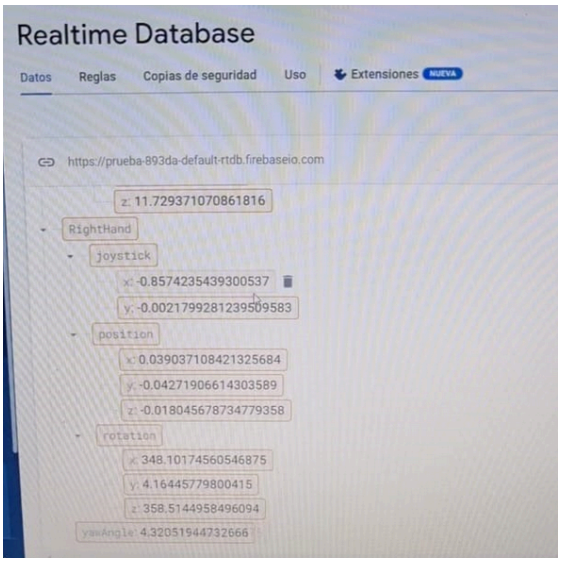


Figura 14. Actualización de Datos a tiempo real Firebase

Como se puede observar en las imágenes, la recopilación y el envío de datos al firebase funciona correctamente, solo depende de la estabilidad de la red a la que esté conectada la computadora y el robot, para brindar al operador un mejor control y fluidez en los movimientos que realice.
Por otro lado, si el operador posiciona los controles fuera del rango de movimientos del robot, este no tendrá respuesta a dicha posición al estar fuera de sus límites.

Conclusiones

La conexión a través de la red cumple el funcionamiento del robot; sin embargo, al ser una red de un proveedor externo tiende a tener saturación en ciertas zonas y ciertos horarios lo que genera retrasos en el censo y la lectura de datos, por lo que puede verse afectado los movimientos del robot respecto al operario y provocar mareos o confusiones al momento de la teleoperación. La base móvil, permite al robot desplazarse en el entorno de forma estable lo que permite una buena maniobrabilidad a distancia.

Bibliografía

1. J. Gottlieb y D. Leech Anderson. "III. teleoperated robots". https://mind.ilstu.edu/curriculum/medical_robotics/teleo.html.

2. Baxter: Redefining robotics and manufacturing (2014). http://www.rethinkrobotics.com/products/baxter. (accedido el 9 de junio de 2023).

3. Cunningham, A., Keddy-Hector, W., Sinha, U., Whalen, D., Kruse, D., Braasch, J., & Wen, J. T. (2014). *Jamster: A mobile dual-arm assistive robot with Jamboxx control. 2014 IEEE International Conference on Automation Science and Engineering (CASE).* doi:10.1109/coase.2014.6899374 (accedido el 9 de junio de 2023).

4. Venator, E., Lee, G. S., & Newman, W. (2013). *Hardware and software architecture of ABBY: An industrial mobile manipulator. 2013 IEEE International Conference on Automation Science and Engineering (CASE).* doi:10.1109/coase.2013.6653969 (accedido el 9 de junio de 2023).

5. Rethink Robotics, 2016, "Baxter Research Robot: Technical Specification Datasheet & Hardware Architecture Overview", http://www.active8robots.com/wp-content/uploads/Baxter-HardwareSpecification-Architecture-Datasheet.pdf. (accedido el 9 de junio de 2023).

6. R.L. Williams II. "Baxter humanoid robot kinematics". https://www.ohio.edu/mechanical-faculty/williams/html/pdf/BaxterKinematics.pdf (accedido el 9 de junio de 2023).

7. Pineau J, Atrash A (2007) SmartWheeler: a robotic wheelchair test-bed for investigating new models of human-robot interaction. In: AAAI spring symposium: multidisciplinary collaboration for socially assistive robotics, pp 59–64 (accedido el 9 de junio de 2023).

8. D. Gajsek. "VR controllers: The way of interacting with the virtual worlds". CIRCUIT STREAM. https://circuitstream.com/blog/vr-controllers-the-way-of-interacting-with-the-virtual-worlds (accedido el 9 de junio de 2023).

9. Meta Quest. "Get started with meta quest development in unity". https://developer.oculus.com/documentation/unity/unity-gs-overview/ (accedido el 9 de junio de 2023).

10. Firebase y Google. "Firebase realtime database". https://firebase.google.com/products/realtime-database?hl=es-419 (accedido el 9 de junio de 2023).

11. RethinkRobotics, "Rethinkrobotics/baxter: Baxter research robot sdk,"Dec 2015. [Online]. Available: https://github.com/RethinkRobotics/baxter (accedido el 9 de junio de 2023).
12. [2] Google, "Agrega firebase a tu proyecto de unity - firebase para unity," May 2019. [Online]. Available:https://firebase.google.com/docs/unity/setup?hl=es-419 (accedido el 9 de junio de 2023).

13. Lu, L., & T. Wen, J. (2017). *Baxter-On-Wheels (BOW): An Assistive Mobile Manipulator for Mobility Impaired Individuals. Trends in Control and Decision-Making for Human–Robot Collaboration Systems, 41–63.* doi:10.1007/978-3-319-40533-9_3 (accedido el 9 de junio de 2023).

14. Lipton, J. I., Fay, A. J., & Rus, D. (2018). Baxter's Homunculus: Virtual Reality Spaces for Teleoperation in Manufacturing. IEEE Robotics and Automation Letters, 3(1), 179–186. doi:10.1109/lra.2017.2737046 (accedido el 9 de junio de 2023).