public float desiredDistanceAhead = 1.0f; // Adjust this value as needed

public float desiredDistanceBehind = 0.5f;

double px2 = 0, py2 = 0, pz2 = 0;

if (dhdGetPosition(ref px2, ref py2, ref pz2, defaultId) >= 0)

{

Vector3 spherePosition = TargetSphere.transform.position;

Vector3 endEffectorPosition = EndEffector.transform.position;

Vector3 sphereToHaptic = endEffectorPosition - spherePosition;

float distance = sphereToHaptic.magnitude;

Vector3 direction = sphereToHaptic.normalized;

// Calculate the repelling force based on the distance

float repellingForce = CalculateRepellingForce(distance);

// Calculate positions for the channel points

Vector3 channelAhead = spherePosition + sphereToHaptic.normalized \* (desiredDistanceAhead);

Vector3 channelBehind = spherePosition - sphereToHaptic.normalized \* (desiredDistanceBehind);

// Calculate the direction vectors for the repelling forces at the channel points

Vector3 directionAhead = (endEffectorPosition - channelAhead).normalized;

Vector3 directionBehind = (endEffectorPosition - channelBehind).normalized;

// Calculate the repelling forces at the channel points

float repellingForceAhead = CalculateRepellingForce(Vector3.Distance(channelAhead, endEffectorPosition));

float repellingForceBehind = CalculateRepellingForce(Vector3.Distance(channelBehind, endEffectorPosition));

// Apply the repelling forces

Vector3 repellingForceVector = -direction \* repellingForce \* repellingForceMultiplier +

directionAhead \* repellingForceAhead \* repellingForceMultiplier +

directionBehind \* repellingForceBehind \* repellingForceMultiplier;

ApplyForceToHapticDevice(repellingForceVector);

}