using System;

using System.Collections.Generic;

using System.Runtime.InteropServices;

using UnityEngine;

using static DLLImportTest;

using System.Collections;

using UnityEngine.ProBuilder.Shapes;

public class DLLImportTest : MonoBehaviour

{

public enum DHD\_STATUS\_ENUM

{

DHD\_STATUS\_POWER,

DHD\_STATUS\_CONNECTED,

DHD\_STATUS\_STARTED,

DHD\_STATUS\_RESET,

DHD\_STATUS\_IDLE,

DHD\_STATUS\_FORCE,

DHD\_STATUS\_BRAKE,

DHD\_STATUS\_TORQUE,

DHD\_STATUS\_WRIST\_DETECTED,

DHD\_STATUS\_ERROR,

DHD\_STATUS\_GRAVITY,

DHD\_STATUS\_TIMEGUARD,

DHD\_STATUS\_WRIST\_INIT,

DHD\_STATUS\_REDUNDANCY,

DHD\_STATUS\_FORCE\_OFF\_CAUSE,

DHD\_STATUS\_LOCKS,

DHD\_STATUS\_AXIS\_CHECKED

}

private bool forcesOn = false;

public double forceTest = 1.0;

public enum deviceStatus { DELTA\_OPEN, DELTA\_CLOSED };

public Transform targetTransform;

public deviceStatus DeviceStatus = deviceStatus.DELTA\_CLOSED;

public GameObject EndEffector;

public GameObject TargetSphere;

public float distanceThreshold = 1.8f;

//variables for the regular attractive force with applying resistive force

private float areaSize = 5.0f; // The size of the area in all three dimensions (5x5x5)

public float timeInsideSphereForForce = 5.0f; // Time user is inside the sphere before resistive force is applied

public float constantForceDuration = 5.0f; // time for constant resistive force

public float maxRandomForceDurationLimit = 10.0f; // Maximum duration for the random force

private float timeInsideTargetSphere = 0.0f;

private float timeSinceRandomForceStart = 0.0f;

private bool applyingRandomForce = false;

private bool applyingRegularForce = true;

private bool isRandomForceOn = false;

//Variables for repelling force

public float repelForceTest = 1.0f;

private float repellingForceMultiplier = 1.0f;

public float maxRepellingforceDistance = 2.0f; // The distance where repelling force is at its maximum

private bool repellingForceOn = false;

//Variables for non linear movement

public float speed = 2f;

private Vector3 targetPosition;

private Vector3 initialPosition;

private float startTime;

private float journeyLength;

private bool moving = false;

private int xyz; // Change the type to int to correctly select the axis

private float amplitude;

private float frequency;

public GameObject spherePrefab;

public Vector3 DhdPosition = Vector3.zero;

IntPtr defaultId = new IntPtr(1);

const int DHD\_MAX\_STATUS = 17;

int[] DHD\_STATUS\_RESULT = new int[DHD\_MAX\_STATUS];

public List<DHD\_STATUS\_ENUM> dhdStatus = new List<DHD\_STATUS\_ENUM>();

[DllImport("dhd64.dll")]

extern static int dhdOpen();

[DllImport("dhd64.dll")]

extern static int dhdStop(IntPtr id);

[DllImport("dhd64.dll")]

extern static int dhdClose(IntPtr id);

[DllImport("dhd64.dll")]

extern static int dhdGetStatus(int[] dhdStatus, IntPtr id);

[DllImport("dhd64.dll")]

extern static int dhdSetBrakes(int val, IntPtr id);

[DllImport("dhd64.dll")]

extern static IntPtr dhdErrorGetLastStr();

[DllImport("dhd64.dll")]

extern static void dhdSleep(double sec);

[DllImport("dhd64.dll")]

extern static int dhdGetPosition(ref double px, ref double py, ref double pz, IntPtr id);

[DllImport("dhd64.dll")]

extern static int dhdEnableForce(UIntPtr val, IntPtr id);

[DllImport("dhd64.dll")]

extern static int dhdSetStandardGravity(double g, IntPtr id);

[DllImport("dhd64.dll")]

extern static int dhdSetForce(double fx, double fy, double fz, IntPtr id);

// Pooled list of spheres

GameObject[] spheres = new GameObject[30];

// Start is called before the first frame update

void Start()

{

Debug.Log("DLLImportTest Start method called");

DhdOpen();

if (dhdEnableForce(new UIntPtr(1), defaultId) >= 0)

{

Debug.Log("Forces set to on");

forcesOn = true;

}

else

{

Debug.LogError("ERROR SETTING FORCES TO ON");

}

// Instantiate your spheres once

for (int i = 0; i < spheres.Length; i++)

{

spheres[i] = Instantiate(spherePrefab, Vector3.zero, Quaternion.identity);

spheres[i].SetActive(false);

}

UpdateDHDStatus();

initialPosition = TargetSphere.transform.position;

GenerateRandomTarget();

GenerateSpheresAlongPath();

}

// Update is called once per frame

void Update()

{

Debug.Log("DLLImportTest Update method called");

if (Input.GetKeyDown(KeyCode.C))

{

DhdClose();

UpdateDHDStatus();

}

if (Input.GetKeyDown(KeyCode.O))

{

DhdOpen();

}

if (Input.GetKeyDown(KeyCode.A))

{

GetDHDPosition();

}

if (Input.GetKeyDown(KeyCode.B))

{

SetDHDBrake(false);

}

if (Input.GetKeyDown(KeyCode.V))

{

SetDHDBrake(true);

}

if (Input.GetKeyDown(KeyCode.S))

{

UpdateDHDStatus();

}

if (Input.GetKeyDown(KeyCode.E))

{

GetLastDHDError();

}

if (Input.GetKeyDown(KeyCode.G))

{

SetDHDGravity(0.0);

}

if (Input.GetKeyDown(KeyCode.F))

{

//ApplyForceTest(true, new Vector3((float)forceTest, (float)forceTest, (float)forceTest));

}

if (Input.GetKeyDown(KeyCode.R))

{

//ApplyForceTest(false, new Vector3((float)0.0f, (float)0.0f, (float)0.0f));

}

sineMovement();

}

private void FixedUpdate()

{

if (DeviceStatus == deviceStatus.DELTA\_OPEN)

{

if (GetDHDPosition() >= 0) //check to see if information from haptic device successful

{

//EndEffector is unity object, this matches the position of haptic device (DhdPosition)

EndEffector.transform.position = DhdPosition;

}

// Method to scale the EndEffctors movment in unity

double px1 = 0, py1 = 0, pz1 = 0;

if (dhdGetPosition(ref px1, ref py1, ref pz1, defaultId) >= 0)

{

Vector3 scaledHapticPosition = new Vector3((float)(px1 \* 100), (float)(pz1 \* 100), (float)(py1 \* 100));

EndEffector.transform.position = scaledHapticPosition;

}

if (forcesOn)

{

//ApplyAttractiveForce(); //to target sphere

//ApplyRepellingForceToTarget(); //to target sphere

//applying forces to the forceSpheres

foreach (GameObject sphere in spheres)

{

if (Vector3.Distance(TargetSphere.transform.position, sphere.transform.position) <= 2.0f)

{

ApplyRepellingForceToSpheres(sphere);

}

}

}

}

}

private void ApplyAttractiveForce()

{

double px = 0, py = 0, pz = 0;

if (dhdGetPosition(ref px, ref py, ref pz, defaultId) >= 0)

{

Vector3 heading = TargetSphere.transform.position - EndEffector.transform.position;

float distance = heading.magnitude;

Vector3 direction = heading / distance;

Vector3 force = new Vector3((float)direction.x \* (float)forceTest, (float)direction.y \* (float)forceTest, (float)direction.z \* (float)forceTest);

if (distance < distanceThreshold)

{

ApplyForceToHapticDevice(Vector3.zero);

}

else

{

ApplyForceToHapticDevice(force);

}

}

}

private void ApplyRepellingForceToTarget()

{

double px = 0, py = 0, pz = 0;

if (dhdGetPosition(ref px, ref py, ref pz, defaultId) >= 0)

{

Vector3 heading = TargetSphere.transform.position - EndEffector.transform.position;

float distance = heading.magnitude;

Vector3 direction = heading.normalized;

float repellingForce = CalculateRepellingForce(distance);

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

ApplyForceToHapticDevice(repellingForceVector);

}

}

private void ApplyRepellingForceToSpheres(GameObject sphere)

{

// Calculate repelling force based on distance from forceSphere

//float repellingForce = CalculateRepellingForceFromForceSphere();

Vector3 heading = sphere.transform.position - EndEffector.transform.position;

float distance = heading.magnitude;

Vector3 direction = heading.normalized;

// Calculate the repelling force based on the distance

float repellingForce = CalculateRepellingForce(distance);

// Apply the force in the opposite direction of the forceSphere

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

// Apply the repelling force to the haptic device

ApplyForceToHapticDevice(repellingForceVector);

}

private void ApplyForceToHapticDevice(Vector3 force)

{

dhdSetForce(force.x, force.z, force.y, defaultId);

}

private float CalculateRepellingForce(float distance)

{

float maxDistanceCalc = maxRepellingforceDistance \* distanceThreshold;

float clampedDistance = Mathf.Clamp(distance, distanceThreshold, maxDistanceCalc);

float normalizedForce = 1.0f - (clampedDistance - distanceThreshold) / (maxDistanceCalc - distanceThreshold);

float repellingForce = normalizedForce \* (float)repelForceTest;

return repellingForce;

}

private Vector3 CatmullRom(Vector3 p0, Vector3 p1, Vector3 p2, Vector3 p3, float t)

{

float t2 = t \* t;

float t3 = t2 \* t;

Vector3 v0 = (p2 - p0) \* 0.5f;

Vector3 v1 = (p3 - p1) \* 0.5f;

return (2 \* p1 - 2 \* p2 + v0 + v1) \* t3 + (-3 \* p1 + 3 \* p2 - 2 \* v0 - v1) \* t2 + v0 \* t + p1;

}

private void sineMovement()

{

if (moving)

{

float distanceCovered = (Time.time - startTime) \* speed;

float t = distanceCovered / journeyLength;

if (t >= 1f)

{

moving = false;

initialPosition = targetPosition; // Set new initial position

DeleteForceSpheresFromList();

GenerateRandomTarget();

GenerateSpheresAlongPath();

}

else

{

Vector3 newPosition = CatmullRom(

initialPosition,

initialPosition,

targetPosition,

targetPosition,

t

);

if (xyz == 0)

{

newPosition.x += Mathf.Sin(t \* Mathf.PI \* 2 \* frequency) \* amplitude;

}

else if (xyz == 1)

{

newPosition.y += Mathf.Sin(t \* Mathf.PI \* 2 \* frequency) \* amplitude;

}

else

{

newPosition.z += Mathf.Sin(t \* Mathf.PI \* 2 \* frequency) \* amplitude;

}

// Apply position constraints

newPosition.x = Mathf.Clamp(newPosition.x, -10f, 10f);

newPosition.y = Mathf.Clamp(newPosition.y, 0f, 10f);

newPosition.z = Mathf.Clamp(newPosition.z, -10f, 10f);

TargetSphere.transform.position = newPosition;

}

}

}

private void GenerateRandomTarget()

{

Vector3 newTarget;

do

{

newTarget = initialPosition + new Vector3(

UnityEngine.Random.Range(-10, 10),

UnityEngine.Random.Range(0, 10),

UnityEngine.Random.Range(-10, 10)

);

if (newTarget.x > 9 || newTarget.y > 9 || newTarget.z > 9)

{

newTarget = new Vector3(0f, 2f, 0f);

}

if (newTarget.x < -9 || newTarget.y < 0 || newTarget.z < -9)

{

newTarget = new Vector3(0f, 5f, 0f);

}

} while (Vector3.Distance(initialPosition, newTarget) < 5f); // find new target that has x distance away from previous target

targetPosition = newTarget;

startTime = Time.time;

journeyLength = Vector3.Distance(transform.position, targetPosition);

moving = true;

xyz = UnityEngine.Random.Range(0, 3);

amplitude = UnityEngine.Random.Range(1f, 5.0f);

frequency = UnityEngine.Random.Range(0.5f, 3.0f);

}

private void GenerateSpheresAlongPath()

{

float totalDistance = Vector3.Distance(initialPosition, targetPosition);

int numSpheres = Mathf.FloorToInt(totalDistance / 0.7f);

for (int i = 0; i <= numSpheres; i++)

{

float journeyFraction = (float)i / numSpheres;

Vector3 spherePosition = Vector3.Lerp(initialPosition, targetPosition, journeyFraction);

if (xyz == 0)

{

spherePosition.x += Mathf.Sin(journeyFraction \* Mathf.PI \* 2 \* frequency) \* amplitude;

}

else if (xyz == 1)

{

spherePosition.y += Mathf.Sin(journeyFraction \* Mathf.PI \* 2 \* frequency) \* amplitude;

}

else

{

spherePosition.z += Mathf.Sin(journeyFraction \* Mathf.PI \* 2 \* frequency) \* amplitude;

}

if (i < 20)

{

spheres[i].transform.position = spherePosition;

spheres[i].SetActive(true);

}

}

}

public void DeleteForceSpheresFromList()

{

for (int i = 0; i < spheres.Length; i++)

{

spheres[i].SetActive(false);

}

}

private bool IsTargetInRangeOfForceSphere(GameObject forceSphere)

{

float distanceToForceSphere = Vector3.Distance(TargetSphere.transform.position, forceSphere.transform.position);

return distanceToForceSphere <= 1.0f;

}

private bool IsEndEffectorNearForceSphere(GameObject forceSphere)

{

float distanceToEndEffector = Vector3.Distance(EndEffector.transform.position, forceSphere.transform.position);

return distanceToEndEffector <= maxRepellingforceDistance; // Set your desired threshold value here

}

public int DhdOpen()

{

// open the first available device

if (dhdOpen() < 0)

{

DeviceStatus = deviceStatus.DELTA\_CLOSED;

IntPtr intPtr = dhdErrorGetLastStr();

string myErrorString = Marshal.PtrToStringAnsi(intPtr);

Debug.LogError(String.Format("error: cannot open device {0}\n", myErrorString));

dhdSleep(2.0);

return -1;

}

else

{

Debug.Log(String.Format("Device Succesfully Opened"));

DeviceStatus = deviceStatus.DELTA\_OPEN;

UpdateDHDStatus();

return 0;

}

}

private void OnDestroy()

{

DhdClose();

}

public int DhdClose()

{

if (dhdClose(defaultId) < 0)

{

IntPtr intPtr = dhdErrorGetLastStr();

string myErrorString = Marshal.PtrToStringAnsi(intPtr);

Debug.LogError(String.Format("error: Failed to stop device {0}\n", myErrorString));

return -1;

}

else

{

DeviceStatus = deviceStatus.DELTA\_CLOSED;

Debug.Log(String.Format("Device Closed!"));

return 0;

}

}

private int UpdateDHDStatus()

{

if (dhdGetStatus(DHD\_STATUS\_RESULT, defaultId) < 0)

{

dhdStatus.Clear();

return -1;

}

else

{

//Debug.Log(String.Format("Succesfully Got Status"));

dhdStatus.Clear();

for (int i = 0; i < DHD\_MAX\_STATUS; i++)

{

int currentResult = DHD\_STATUS\_RESULT[i];

if (currentResult > 0)

{

dhdStatus.Add((DHD\_STATUS\_ENUM)i);

}

}

return 0;

}

}

public int GetDHDPosition()

{

double px = 0;

double py = 0;

double pz = 0;

if (dhdGetPosition(ref px, ref py, ref pz, defaultId) < 0)

{

Debug.LogError("ERROR GETTING POSITION");

return -1;

}

else

{

//Debug.Log(String.Format("{0:0.000},{1:0.000},{2:0.000}", px, py, pz));

DhdPosition = new Vector3((float)px, (float)pz, (float)py);

return 0;

}

}

public int SetDHDBrake(bool brakeOn)

{

int success = -1;

if (brakeOn)

{

dhdEnableForce(new UIntPtr(1), defaultId);

if (dhdSetBrakes(1, defaultId) < 0)

{

Debug.LogError("ERROR TURNING ON BRAKE");

success = -1;

}

else

{

success = 0;

}

}

else

{

dhdEnableForce(new UIntPtr(0), defaultId);

if (dhdSetBrakes(0, defaultId) < 0)

{

Debug.LogError("ERROR TURNING OFF BRAKE");

success = -1;

}

else

{

success = 0;

}

}

UpdateDHDStatus();

return success;

}

public void GetLastDHDError()

{

IntPtr intPtr = dhdErrorGetLastStr();

string myErrorString = Marshal.PtrToStringAnsi(intPtr);

Debug.LogError(String.Format("Last Error: {0}\n", myErrorString));

}

public void SetDHDGravity(double g)

{

dhdSetStandardGravity(g, defaultId);

}

}