REPUBLIK INDONESIA
KEMENTERIAN HUKUM DAN HAK ASASI MANUSIA

SURAT PENCATATAN CIPTAAN

Dalam rangka pelindungan ciptaan di bidang ilmu pengetahuan, seni dan sastra berdasarkan Undang-Undang Nomor 28 Tahun 2014 tentang Hak Cipta, dengan ini menerangkan:

Nomor dan tanggal permohonan	:	EC00202411536, 1 Februari 2024
Pencipta		
Nama	:	**Muhammad Umar Shalih, Jeremy Jonathan dkk**
Alamat	:	Jl. Pondok Hijau VI/23, RT006 RW013, Pondok Pinang, Kebayoran Lama, Jakarta Selatan, Kebayoran Lama, Jakarta Selatan, DKI Jakarta, 12310
Kewarganegaraan	:	Indonesia
Pemegang Hak Cipta		
Nama	:	**Muhammad Umar Shalih, Jeremy Jonathan dkk**
Alamat	:	Jl. Pondok Hijau VI/23, RT006 RW013, Pondok Pinang, Kebayoran Lama, Jakarta Selatan, Kebayoran Lama, Jakarta Selatan, DKI Jakarta, 12310
Kewarganegaraan	:	Indonesia
Jenis Ciptaan	:	**Program Komputer**
Judul Ciptaan	:	**MOTION TRANSLATOR**
Tanggal dan tempat diumumkan untuk pertama kali di wilayah Indonesia atau di luar wilayah Indonesia	:	1 Februari 2024, di Jakarta Selatan
Jangka waktu pelindungan	:	Berlaku selama 50 (lima puluh) tahun sejak Ciptaan tersebut pertama kali dilakukan Pengumuman.
Nomor pencatatan	:	000586907

adalah benar berdasarkan keterangan yang diberikan oleh Pemohon.
Surat Pencatatan Hak Cipta atau produk Hak terkait ini sesuai dengan Pasal 72 Undang-Undang Nomor 28 Tahun 2014 tentang Hak Cipta.

a.n. MENTERI HUKUM DAN HAK ASASI MANUSIA
DIREKTUR JENDERAL KEKAYAAN INTELEKTUAL
u.b
Direktur Hak Cipta dan Desain Industri

Anggoro Dasananto
NIP. 196412081991031002

Disclaimer:
Dalam hal pemohon memberikan keterangan tidak sesuai dengan surat pernyataan, Menteri berwenang untuk mencabut surat pencatatan permohonan.

LAMPIRAN PENCIPTA

No	Nama	Alamat
1	Muhammad Umar Shalih	Jl. Pondok Hijau VI/23, RT006 RW013, Pondok Pinang, Kebayoran Lama, Jakarta Selatan, Kebayoran Lama, Jakarta Selatan
2	Jeremy Jonathan	Perum Aster Blok B-3 No. 32, RT006 RW005, Cibodas, Cibodas, Kota Tangerang, Cibodas, Tangerang
3	Teja Endra Eng Tju	Alam Sutera Buana III No. 12 A, RT001 RW009, Pakulonan, Serpong Utara, Tangerang Selatan, Serpong Utara, Tangerang Selatan

LAMPIRAN PEMEGANG

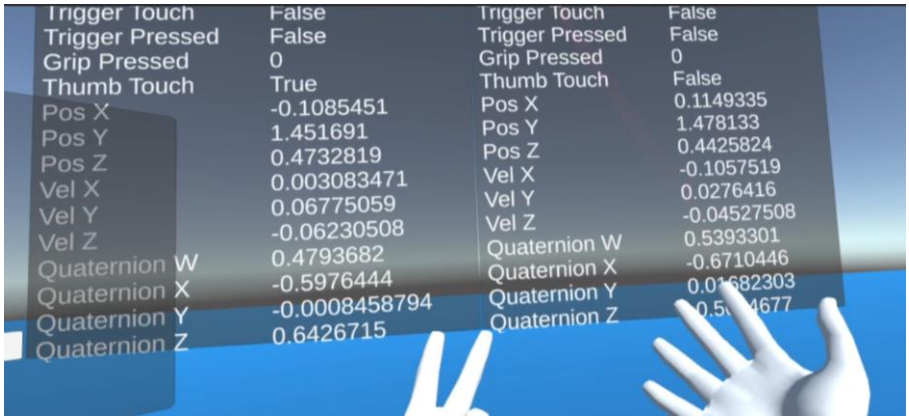
No	Nama	Alamat
1	Muhammad Umar Shalih	Jl. Pondok Hijau VI/23, RT006 RW013, Pondok Pinang, Kebayoran Lama, Jakarta Selatan, Kebayoran Lama, Jakarta Selatan
2	Jeremy Jonathan	Perum Aster Blok B-3 No. 32, RT006 RW005, Cibodas, Cibodas, Kota Tangerang, Cibodas, Tangerang
3	Teja Endra Eng Tju	Alam Sutera Buana III No. 12 A, RT001 RW009, Pakulonan, Serpong Utara, Tangerang Selatan, Serpong Utara, Tangerang Selatan

MOTION TRANSLATOR

Aplikasi ini dibangun dengan menggunakan Unity Editor, suatu aplikasi antarmuka pengguna yang terhubung dengan perangkat Virtual Reality Meta Quest 2, sebagai media perantara untuk merekam data gerakan isyarat tangan. Tampilan pada headset Meta Quest 2 ditunjukkan pada Gambar 1.

Gambar 1. Antarmuka Pengguna

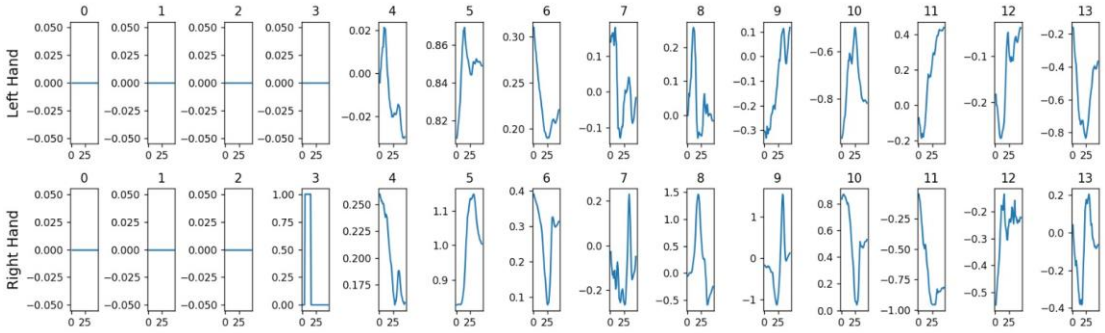
Sensor yang diterapkan teridiri dari 28 parameter masukan (tangan kiri dan kanan) yang berubah-ubah pada saat dilakukan gerakan isyarat tangan, seperti tampak pada Gambar 2.



Gambar 2. Parameter Isyarat Tangan

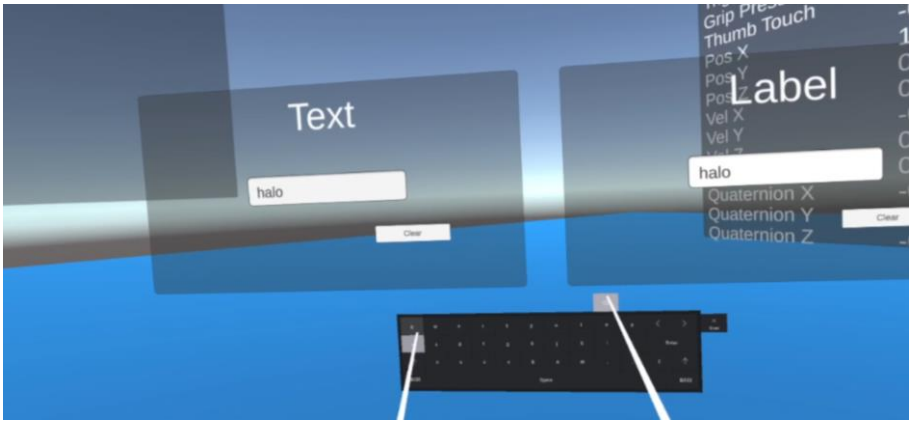
Hasil perekaman data yang berupa grafik seperti contoh pada Gambar 3. Setiap data terdiri dari 14 parameter dari tangan kiri dan 14 parameter lagi dari tangan kanan dengan parameter *Trigger Touch, Trigger Pressed, Grip Pressed, Thumb Touch, Position* (X, Y, Z), *Velocity* (X, Y, Z), *Quaternion* (W, X, Y, Z).

Dalam kode program ditambahkan formula sehingga dapat dipastikan bahwa data yang terkumpul bersifat relatif terhadap headset. Pemadanan ini bertujuan untuk mencegah gerakan tangan memengaruhi rotasi tubuh, dengan jelas mencerminkan arah wajah pengguna tanpa menyebabkan interferensi. Proses ini melibatkan modifikasi lokasi titik pusat dan rotasi global berdasarkan orientasi *headset*.



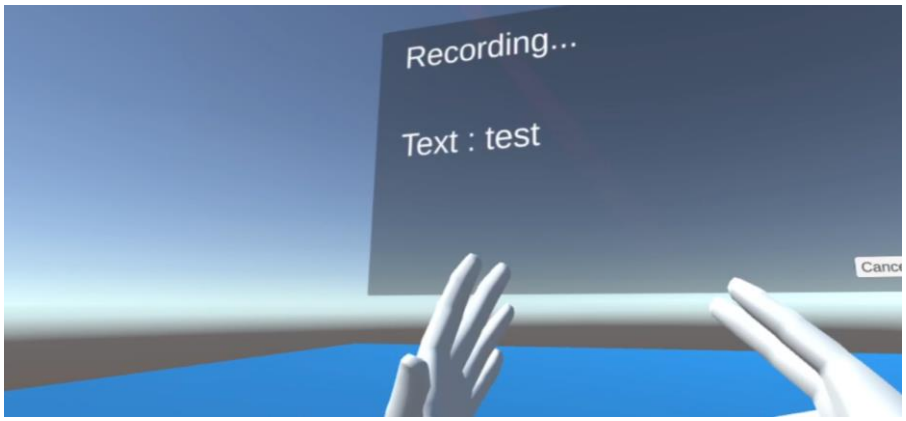
Gambar 3. Contoh Hasil Perekaman Data

Aplikasi dilengkapi fasilitas isian teks dan memberikan label pada data yang akan direkam, disajikan pada Gambar 4.



Gambar 4. Isian Teks untuk Label Data

Pada Gambar 5 tampak proses perekaman data dengan melakukan gerakan isyarat tertentu sesuai dengan keperluan.



Gambar 5. Proses Perekaman Data

Berikut ini adalah kode utama program.

```
using System;
using System.Collections;
using System.Net.Http;
using System.Collections.Generic;
using UnityEngine;
```

```
using UnityEngine.XR;
using TMPro;
using Newtonsoft.Json;
using UnityEngine.UI;
using Newtonsoft.Json.Linq;

namespace MotionTranslator {

    [RequireComponent(typeof(VRInput))]
    public class RecordingStatus : MonoBehaviour
    {

        public TMP_InputField textField;
        public TMP_InputField labelField;
        public TextMeshProUGUI content;
        public TextMeshProUGUI countClassText;
        public TextMeshProUGUI recordingText;
        public TextMeshProUGUI statusText;
        public TextMeshProUGUI dataCountText;
        public Button cancelButton;
        public Button sendToNetworkButton;
        public Button popLastElementButton;

        private List<List<float>> _tempLeftControllerRecordData;
        private List<List<float>> _tempRightControllerRecordData;
        private VRInput _controller;
        private bool _recording;
        private bool _prevButtonState;
        private bool _cancelling;

        private List<object> _dataToSendThroughNetwork;

        private String _previousClassValue = "";
        private String _currentClassValue = "";
        private int _countClassValue = 0;



        // Start is called before the first frame update
        void Start()
        {
            _recording = false;
            _cancelling = false;
            _prevButtonState = false;
            _tempLeftControllerRecordData = new List<List<float>>();
            _tempRightControllerRecordData = new List<List<float>>();

            _dataToSendThroughNetwork = new List<object>();

            _controller = GetComponent<VRInput>();
```

```
            if (cancelButton == null)
            {
                cancelButton = GetComponent<Button>();
            }

            initializeListener();
        }

        // Update is called once per frame
        void Update()
        {
            if(_controller._rightController.isValid){
                CheckRecordingButton();
            }

            if(_recording){
                ShowRecordingMessage();
                RecordControllerData();
                ShowCancelButton();
            }else{
                if((_tempLeftControllerRecordData.Count > 0 &&
_tempRightControllerRecordData.Count > 0) && !_cancelling){
                    addRecordControllerDataToList();
                    _tempLeftControllerRecordData.Clear();
                    _tempRightControllerRecordData.Clear();
                    ShowClassCount();
                }

                if(_dataToSendThroughNetwork.Count > 0)
                {
                    ShowSendToNetworkButtonAndCountMessage();
                }
                else
                {
                    HideSendToNetworkButtonAndCountMessage();
                }
                ShowNotRecordingMessage();
                HideCancelButton();
            }
        }

        void initializeListener()
        {
            cancelButton.onClick.AddListener(HandleCancleButtonClick);
            sendToNetworkButton.onClick.AddListener(HandleSendThroughNetwork);
            popLastElementButton.onClick.AddListener(HandlePopLastElement);
        }

        void HandleCancleButtonClick()
        {
```

```
            _cancelling = true;
            _recording = false;
            _tempLeftControllerRecordData.Clear();
            _tempRightControllerRecordData.Clear();
            statusText.text = "Record Cancelled.";
            _cancelling = false;
        }

        void HandlePopLastElement()
        {
            if (_dataToSendThroughNetwork.Count > 0)
            {

_dataToSendThroughNetwork.RemoveAt(_dataToSendThroughNetwork.Count - 1);
// Remove the last element
            }
        }

        void ShowCancelButton()
        {
            cancelButton.gameObject.SetActive(true);
        }

        void HideCancelButton()
        {
            cancelButton.gameObject.SetActive(false);
        }

        void ShowSendToNetworkButtonAndCountMessage()
        {
            sendToNetworkButton.gameObject.SetActive(true);
            dataCountText.text = "Data Count : " +
_dataToSendThroughNetwork.Count.ToString();
        }

        void HideSendToNetworkButtonAndCountMessage()
        {
            sendToNetworkButton.gameObject.SetActive(false);
            dataCountText.text = "";
        }

        void CheckRecordingButton()
        {
            bool currentButtonState = _controller.rightController.getSecondaryButton();
            if (currentButtonState && !_prevButtonState){
                // Flip switch
                _recording = !_recording;
            }

            _prevButtonState = currentButtonState;
```

```
    }

    void ShowRecordingMessage()
    {
        recordingText.text = "Recording...";
        content.text = "Text : " + textField.text;
    }

    void ShowNotRecordingMessage()
    {
        recordingText.text = "Not Recording...";
        content.text = "Text : ";
    }

    void ShowClassCount()
    {
        _currentClassValue = labelField.text;
        if (_currentClassValue != _previousClassValue)
        {
            _previousClassValue = _currentClassValue;
            _countClassValue = 1;
        }
        else
        {
            _countClassValue++;
        }

        countClassText.text = _currentClassValue + " : " +
_countClassValue.ToString();
    }

    void RecordControllerData()
    {
        Controller vrHeadset = _controller.HMD;

        // Get the headset's position and rotation
        Vector3 headsetPosition = vrHeadset.getPosition();
        Quaternion headsetRotation = vrHeadset.getRotation();
        Vector3 headsetVelocity = vrHeadset.getVelocity();

        // Get the positions, velocities, and rotations of the left and right controllers
        Vector3 leftPosition = _controller.leftController.getPosition();
        Vector3 leftVelocity = _controller.leftController.getVelocity();
        Quaternion leftQuaternion = _controller.leftController.getRotation();

        Vector3 rightPosition = _controller.rightController.getPosition();
        Vector3 rightVelocity = _controller.rightController.getVelocity();
        Quaternion rightQuaternion = _controller.rightController.getRotation();

        // Transform controller data relative to headset
```

```
        Vector3 transformedLeftPosition = headsetRotation * (leftPosition - headsetPosition);
        Vector3 transformedRightPosition = headsetRotation * (rightPosition - headsetPosition);

        Quaternion transformedLeftRotation = Quaternion.Inverse(headsetRotation) * leftQuaternion;
        Quaternion transformedRightRotation = Quaternion.Inverse(headsetRotation) * rightQuaternion;

        Vector3 transformedLeftVelocity = leftVelocity - headsetVelocity;
        Vector3 transformedRightVelocity = rightVelocity - headsetVelocity;

        _tempLeftControllerRecordData.Add(new List<float> {
            _controller.leftController.getTriggerTouch() ? 1.0f : 0.0f,
            _controller.leftController.getTriggerButton() ? 1.0f : 0.0f,
            _controller.leftController.getGrip(),
            _controller.leftController.getThumbTouch() ? 1.0f : 0.0f,
            transformedLeftPosition.x,
            transformedLeftPosition.y,
            transformedLeftPosition.z,
            transformedLeftVelocity.x,
            transformedLeftVelocity.y,
            transformedLeftVelocity.z,
            transformedLeftRotation.w,
            transformedLeftRotation.x,
            transformedLeftRotation.y,
            transformedLeftRotation.z
        });

        _tempRightControllerRecordData.Add(new List<float> {
            _controller.rightController.getTriggerTouch() ? 1.0f : 0.0f,
            _controller.rightController.getTriggerButton() ? 1.0f : 0.0f,
            _controller.rightController.getGrip(),
            _controller.rightController.getThumbTouch() ? 1.0f : 0.0f,
            transformedRightPosition.x,
            transformedRightPosition.y,
            transformedRightPosition.z,
            transformedRightVelocity.x,
            transformedRightVelocity.y,
            transformedRightVelocity.z,
            transformedRightRotation.w,
            transformedRightRotation.x,
            transformedRightRotation.y,
            transformedRightRotation.z
        });
    }

    void addRecordControllerDataToList()
    {
```

```
        List < List<float> > _leftControllerRecordData = new
List<List<float>>(_tempLeftControllerRecordData);
        List < List<float> > _rightControllerRecordData = new
List<List<float>>(_tempRightControllerRecordData);
        var jsonObject = new
        {
            table = "data_basic_asl",
            classValue = labelField.text,
            text = textField.text,
            data = new
            {
                _leftControllerRecordData,
                _rightControllerRecordData
            }
        };

        _dataToSendThroughNetwork.Add(jsonObject);
    }

    void HandleSendThroughNetwork()
    {
        statusText.text = "Sending Record Data Through Network...";

        var jsonSender = new JsonSender();

        string url = "http://127.0.0.1:5000/api/store";

        try{
            HttpResponseMessage response = jsonSender.SendJson(url,
_dataToSendThroughNetwork);

            if(response == null){
                statusText.text = "Failed to send JSON data";
                return;
            }

            if(response.IsSuccessStatusCode){
                statusText.text = "Record has been sent successfully!";
                _dataToSendThroughNetwork.Clear();
            }else{
                statusText.text = "Failed to send JSON data";
            }
        }catch(Exception ex){
            statusText.text = $"Error sending JSON data : {ex.Message}";
        }
    }
  }
}
```