BAB 3
METODOLOGI PENELITIAN

3.1 Metodologi Penelitian

Dalam penelitian ini, diimplementasikan beberapa metodologi dan perancangan sistem yang dibuat dengan Python dan Unity serta pengumpulan dataset yang nantinya digunakan untuk membuat model *Hand Gesture Classification*. Tahap-tahap yang dilaksanakan adalah sebagai berikut:

1. Studi literatur
   Dalam studi literatur, dilakukan pembelajaran dan mendalami topik mengenai MediaPipe, MediaPipe Hand, Tensorflow, Keras, Python, dan Unity. Serta beberapa penelitian sebelumnya yang berkaitan, terutama penelitian yang juga menggunakan MediaPipe Hand untuk melakukan *Hand Gesture Classification* dan juga penelitian yang menggunakan Deep Neural Network untuk membangun model.

2. Analisa Kebutuhan
   Kebutuhan yang dibutuhkan dalam penelitian ini adalah *dataset*, perangkat dengan sistem operasi Windows, software seperti Google Colaboratory, bahasa pemrograman Python beserta Pycharm sebagai IDE, dan bahasa pemrograman C# untuk Unity serta Visual Studio sebagai IDE.

3. Perancangan dan Pembuatan Sistem
   Pada tahap ini, dirancang *flowchart* dari kedua sistem yang utama yaitu, *hand tracking capture* dan *hand tracking receiver*. *hand tracking capture* dibuat dengan menggunakan Python, sistem ini digunakan untuk mengumpulkan dataset, serta menjalankan model untuk mendeteksi *hand gesture*. MediaPipe *hand tracking* dijalankan dengan menggunakan *webcam* sebagai *input*. Kemudian pada *window* aplikasi, tercantum informasi tentang mode dari aplikasi. Aplikasi memiliki mode untuk *hand tracking* biasa, dan mode untuk merekam data. Selain untuk tracking, aplikasi Python juga digunakan untuk menjalankan inferensi *classification*. Data disimpan dalam bentuk JavaScript Object Notation (JSON) dan dikirim menggunakan protokol komunikasi User Datagram Protocol (UDP) secara *local* ke Unity. Sistem *hand tracking receiver* dibuat dengan menggunakan Unity, disini data JSON dari *hand*

tracking capture diterima dan digunakan untuk membuat sebuah tangan virtual.

4. Pengumpulan Dataset

   Pengumpulan dataset dilakukan dengan menggunakan sistem *hand tracking capture* yang sudah dibuat. Bentuk dari data yang dikumpulkan adalah (64,), dimana indeks ke 1 merupakan label, dan 63 lainnya merupakan 21 koordinat landmark 3D yang sudah dijadikan dalam bentuk *one-dimensional list*. Contoh bentuk data dapat dilihat pada gambar 3.1. Sebelum dikumpulkan, data di*pre-process* terlebih dahulu. Data hasil *pre-processing* kemudian ditambahkan *label* sesuai dengan gesture yang merepresentasikannya dan semua data dikumpulkan ke dalam satu file Comma-separated values (CSV) sebagai *dataset*. *Dataset* tersebut berisi data untuk 10 *gesture* dengan total 3506 data. Terdiri atas $\sim$ 300 data per-*gesture*, dimana $\sim$ 150 merupakan data untuk tangan kiri, dan $\sim$ 300 untuk data tangan kanan.

```
(64,)
[ 0.          0.          0.          0.          0.16413374 -0.12462006
 -0.00607903  0.27355623 -0.26443768 -0.03647416  0.34042552 -0.40729484
 -0.06382979  0.39209726 -0.51975685 -0.09726444  0.16413374 -0.4224924
 -0.10030395  0.20972644 -0.65957445 -0.15197569  0.2218845  -0.7993921
 -0.17325228  0.2218845  -0.90577507 -0.18541034  0.04559271 -0.43465045
 -0.12765957  0.03951368 -0.71428573 -0.18541034  0.02735562 -0.88145894
 -0.21276596  0.01519757 -1.         -0.23100305 -0.07294833 -0.41033435
 -0.14893617 -0.09422492 -0.668693   -0.20364742 -0.10334346 -0.8267477
 -0.23100305 -0.10638298 -0.94224924 -0.24924012 -0.18237083 -0.3586626
 -0.16717325 -0.23100305 -0.5531915  -0.21884498 -0.26139817 -0.67173254
 -0.23708206 -0.27963525 -0.7720365  -0.2462006 ]
```

Gambar 3.1. Gambar contoh bentuk dataset

5. Membuat Model

   Setelah dataset terkumpul, arsitektur dari Dense Neural Network mulai dibuat dengan menggunakan Tensorflow dan Keras. Model dibuat dengan menggunakan Google Colaboratory. Akan dilakukan juga uji coba dengan berbagai skenario untuk mendapatkan performa model yang paling optimal.

6. Implementasi

   Pada tahap ini, rancangan untuk membuat sistem dan model yang sudah dibuat mulai diimplementasikan. *Hand Gesture Capture* yang terdiri atas *hand tracking* dan *hand gesture classification* diimplementasikan pada Python dan *Hand Gesture Receiver* diimplementasikan pada aplikasi 3D Unity.

7. Evaluasi

   Pada tahap ini, model yang sudah dilatih diuji dan dievaluasi dengan metrik penilaian *accuracy*, *precision*, *recall*, dan *f1-score* untuk mengetahui performa dari model yang sudah dilatih agar mengetahui apakah model tersebut perlu diperbaiki atau dikembangkan lagi.

8. Konsultasi dan Penulisan Laporan

   Dalam proses penelitian ini, dilakukan juga konsultasi dengan ahli untuk mengarahkan bagaimana penelitian ini sebaiknya difokuskan. Selain itu, konsultasi juga dilakukan untuk mencari solusi dari masalah yang dihadapi. Penelitian ini juga dibuat lebih fokus ke aspek *hand gesture classification* dibandingkan dengan implementasinya pada aplikasi 3D. Penulisan laporan juga dilakukan secara berkala dan progress dari tiap Bab disampaikan kepada pembimbing untuk kemudian dilakukan konsultasi jika dibutuhkan.

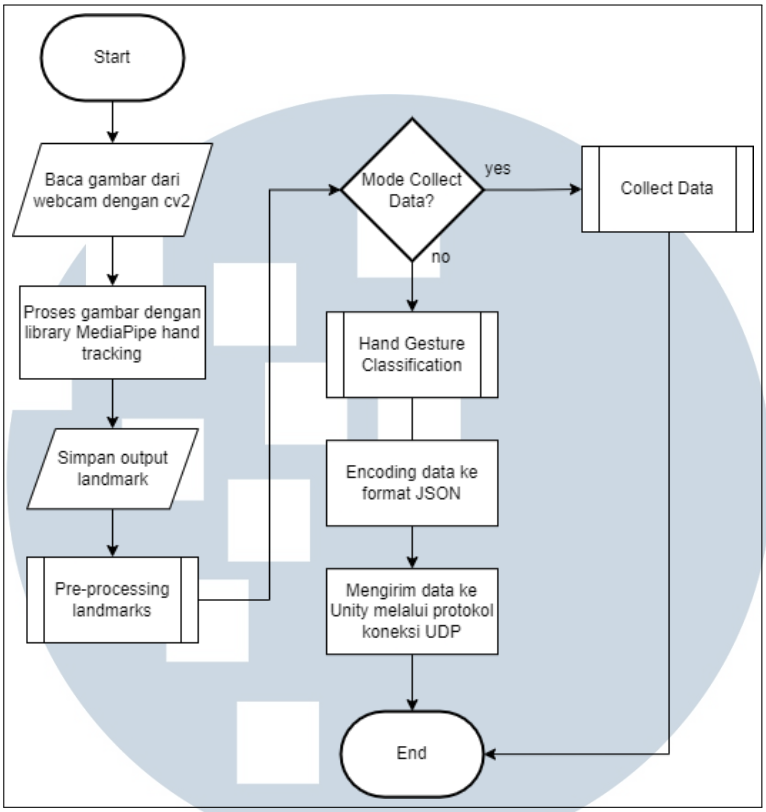
3.2 Perancangan dan Pembuatan Sistem

Terdapat dua sistem yang dibuat, *hand tracking capture* dan *hand tracking receiver*. Sistem *hand tracking capture* dibuat dengan menggunakan Python, sistem ini digunakan untuk mengumpulkan dataset, serta menjalankan model untuk mendeteksi *hand gesture*. MediaPipe *hand tracking* dijalankan dengan menggunakan gambar yang didapatkan dari *webcam* sebagai *input*. Pada *window* aplikasi di sebelah kiri, tercantum informasi tentang mode dari aplikasi. Aplikasi memiliki mode untuk *hand tracking* biasa, dan mode untuk merekam data. Selain untuk tracking, *hand tracking capture* juga digunakan untuk menjalankan inferensi *classification*. Data disimpan dalam bentuk JavaScript Object Notation (JSON) dan dikirim dengan menggunakan protokol koneksi User Datagram Protocol (UDP) secara *local* ke Unity. Sistem *hand tracking receiver* dibuat pada Unity, data JSON dari *hand tracking capture* diterima dan digunakan untuk membuat representasi tangan virtual beserta interaksinya di dalam Unity.

3.2.1 Flowchart Utama Hand Tracking Capture

Gambar 3.2 merupakan *flowchart* utama dari sistem *Hand Tracking Capture*. *Hand Tracking Capture* menggunakan gambar dari kamera untuk melakukan Hand Tracking dengan menggunakan *library* dari MediaPipe. Sebelum semua proses dilakukan, *library* atau *Interpreter* yang dibutuhkan di-*import* terlebih dahulu.

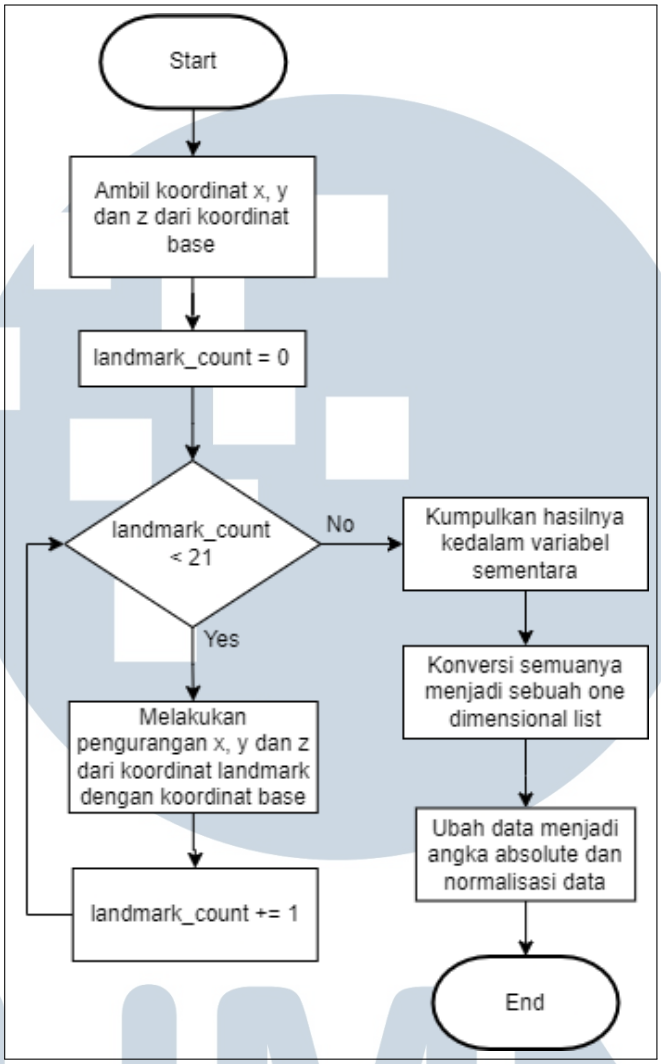
Pertama, OpenCV (cv2) digunakan untuk meng-*capture* gambar dari kamera dan menggunakannya sebagai *input* untuk diproses oleh MediaPipe Hand Tracking. MediaPipe mendeteksi tangan pada gambar tersebut, dan memberikan sebuah *output* yang terdiri dari 21 *landmark* beserta info penting lainnya, seperti tangan yang terdeteksi merupakan tangan kiri atau kanan. Kemudian, data *landmark* di *pre-process* untuk mengkonversi koordinat *landmark* menjadi koordinat yang relatif, koordinat yang memiliki acuan pada satu titik. Koordinat relatif dapat membuat proses pelatihan dan klasifikasi model menjadi lebih efektif. Dengan mengkonversi koordinat *landmark* menjadi koordinat relatif, model dapat menghiraukan masalah koordinat yang berubah secara signifikan berdasarkan dari besar-kecilnya (*scaling*) tangan dan lokasi suatu tangan relatif ke kamera. *Pre-processing* juga dijalankan untuk mengubah bentuk data menjadi *one dimensional list* agar sesuai dengan yang dibutuhkan model. Data *landmark* hasil *pre-processing* ini digunakan untuk Collect Data maupun untuk menjalankan model Hand Gesture Classification, tergantung mode apa yang dipilih oleh pengguna pada aplikasi. Untuk Hand Gesture Classification, model dijalankan pada Python dengan menggunakan sebuah *Interpreter* dari *library* Tensorflow. Semua data kemudian dikumpulkan dan disimpan dalam sebuah variabel *data* dalam bentuk JSON, kemudian di-*encode* menjadi bentuk *string*. Data tersebut memiliki komponen atau elemen: *landmark* yang telah disatukan dan dengan tipe data string (tanpa dilakukan *pre-processsing*), hasil klasifikasi (dalam bentuk string), dan tipe tangan yang terdeteksi (tangan kiri atau kanan). Terakhir, data JSON yang telah di-*encode* dikirimkan ke Unity melalui koneksi UDP secara *local*.



Gambar 3.2. *Flowchart* utama *Hand Tracking Capture* pada Python

A Flowchart Pre-Processing Landmarks

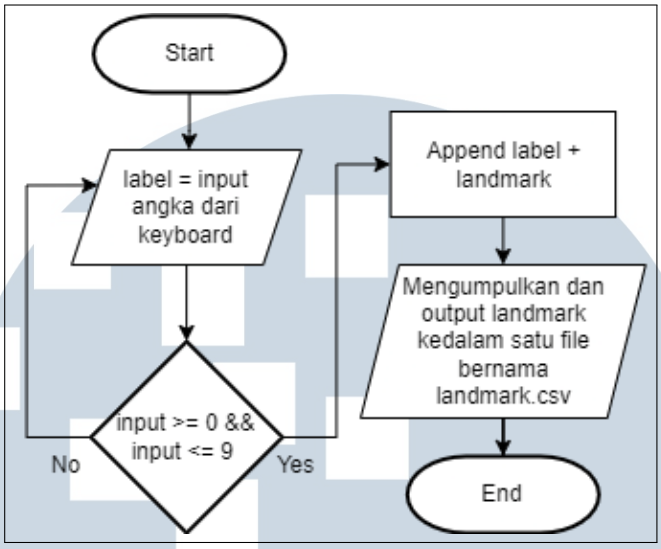
Pre-processing dilakukan pada seluruh 21 titik koordinat *landmark*. Koordinat *landmark* tersebut dikonversi menjadi koordinat relatif dengan menentukan sebuah koordinat yang dapat menjadi acuan atau dengan menentukan sebuah *base coordinate*. Kemudian, nilai *x*, *y*, dan *z* dari *base coordinate* disimpan untuk digunakan sebagai pengurang saat menghitung koordinat relatif dari tiap koordinat *landmark*. Proses *looping* berjalan untuk mengiterasi semua 21 *landmark* dan hasilnya disimpan pada suatu variabel sementara. Pada proses *looping* ini, nilai *x*, *y*, dan *z* dari *landmark* dikurangi dengan nilai $base_x$, $base_y$, dan $base_z$. Setelah proses *looping* selesai, semua data yang ada pada array *tempLmList* dikonversi menjadi *one dimensional list*. Setelah itu, *tempLmList* yang telah berisi koordinat relatif diabsolutkan untuk menghilangkan angka minus, dan dinormalisasi berdasarkan *value* dari nilai tertinggi agar distribusi data menjadi lebih seragam dan memudahkan proses dalam melatih model. Data koordinat *landmark* yang telah dilakukan *Pre-Processing* kemudian diteruskan ke proses selanjutnya. Gambar *flowchart Pre-Processing* dapat dilihat pada Gambar 3.3.



Gambar 3.3. *Flowchart Pre-processing landmarks*

B Flowchart Collect Data

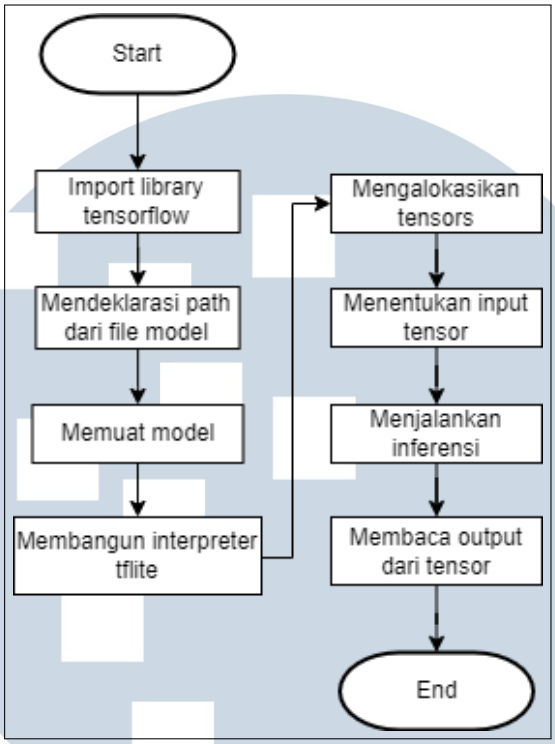
Apabila pengguna memilih untuk *Collect Data*, pengguna dapat memberikan *input* berupa angka, mulai dari angka 0 - 9. Selain dari angka tersebut, aplikasi tidak melakukan apa-apa dan kembali untuk menunggu *input* dari pengguna. Angka yang di-*input* ini menjadi label pada *dataset* yang dibuat. Saat pengguna menekan suatu angka, angka tersebut ditambahkan dipaling depan data *landmark* yang sudah dilakukan *pre-processing*. Data *landmark* yang telah ditambahkan label dikumpulkan dan disimpan menjadi *dataset* ke dalam satu file CSV. Gambar *flowchart Collect Data* dapat dilihat pada Gambar 3.4.



Gambar 3.4. *Flowchart Collect Data*

C Flowchart Hand Gesture Classification

Gambar 3.5 merupakan *flowchart* dari proses *Hand Gesture Classification* yang dilakukan pada Python. Klasifikasi dilakukan dengan menggunakan *library* tensorflow yang ada pada Python. Untuk menjalankan inferensi model menggunakan tensorflow, diperlukan sebuah *Interpreter* yang dibangun sesuai dengan model yang ingin dijalankan. Pertama, *path* dari model perlu didefinisikan terlebih dahulu, *path* digunakan oleh *Interpreter* untuk menemukan dimana file model yang ingin digunakan berada. Setelah model berhasil dimuat, *Interpreter* dapat membaca model untuk mengetahui apa saja kondisi yang diperlukan agar model tersebut dapat melakukan inferensi. *Interpreter* dapat mulai dibangun dengan membaca bagaimana struktur dari model yang dimuat, biasanya sebuah model memiliki *default input* dan *output* yang sesuai dengan strukturnya masing-masing. Setelah struktur *Interpreter* sudah terbentuk sesuai dengan modelnya, *tensors* dapat mulai untuk dialokasikan. *Tensors* ini terdiri dari *input tensor* dan *output tensor*. *Input tensor* adalah data yang ingin kita gunakan untuk melakukan inferensi pada model, dan *output tensor* adalah hasil inferensi-nya. Setelah *Interpreter* menerima suatu *input tensor*, proses inferensi dapat dijalankan. Hasil dari inferensi tersebut tersimpan di dalam *output tensor*, yang dapat digunakan sebagai hasil dari inferensi atau secara spesifik dalam kasus ini, klasifikasi *Hand Gesture* yang dilakukan oleh model.

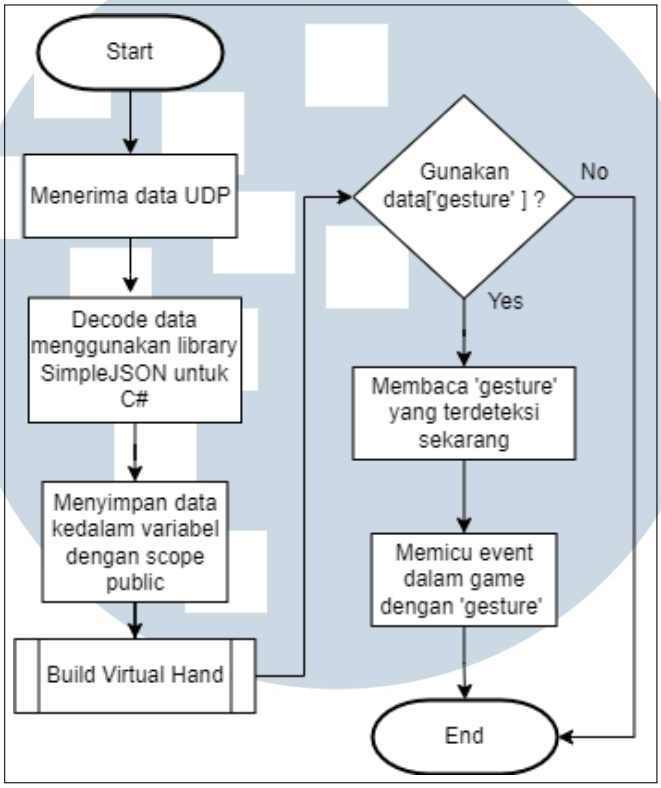


Gambar 3.5. *Flowchart Hand Gesture Classification*

3.2.2 Flowchart Utama Hand Tracking Receiver

Gambar 3.6 merupakan flowchart utama dari *Hand Tracking Receiver*. *Hand Tracking Receiver* menerima data melalui koneksi UDP dari IP Address *local* dan *port* yang sudah ditentukan sebelumnya, yang sama dengan Hand Tracking Capture. Data yang diterima di-*decode* dengan menggunakan *library* bernama SimpleJSON, *library* ini dapat memproses data dengan format JSON dibahasa pemrograman C#. Data tersebut tersimpan di Unity dalam sebuah variabel dengan *scope public*, sehingga dapat digunakan oleh komponen Unity lainnya jika diperlukan. Untuk membuat *virtual hand*, data yang diperlukan hanyalah data *landmarks* saja. Setelah *virtual hand* dibuat, data *gesture* yang didapatkan di dalam JSON tersebut juga dapat diimplementasikan secara bersamaan. *Gesture* yang terdeteksi ini dapat digunakan dalam berbagai kasus, tergantung bagaimana *developer* ingin mengimplementasi *gesture* tersebut. Dengan adanya data *gesture* ini, *virtual hand* yang dibuat dapat kita beri *logic* tambahan, misalnya dengan membuat *virtual hand* hanya dapat berinteraksi dengan sebuah *terminal* atau sebuah *user interface* yang hanya bisa disentuh dengan jari telunjuk saja. *Logic* atau aturan ini dapat dibuat sesuai keinginan *developer* dari aplikasi tersebut. Dengan adanya *gesture*

ini, interaksi dalam *environment* Unity dapat menjadi lebih kompleks karena dapat diperluas ataupun dibatasi dan diatur sedemikian rupa sesuai dengan konsep yang diinginkan oleh *developer*.

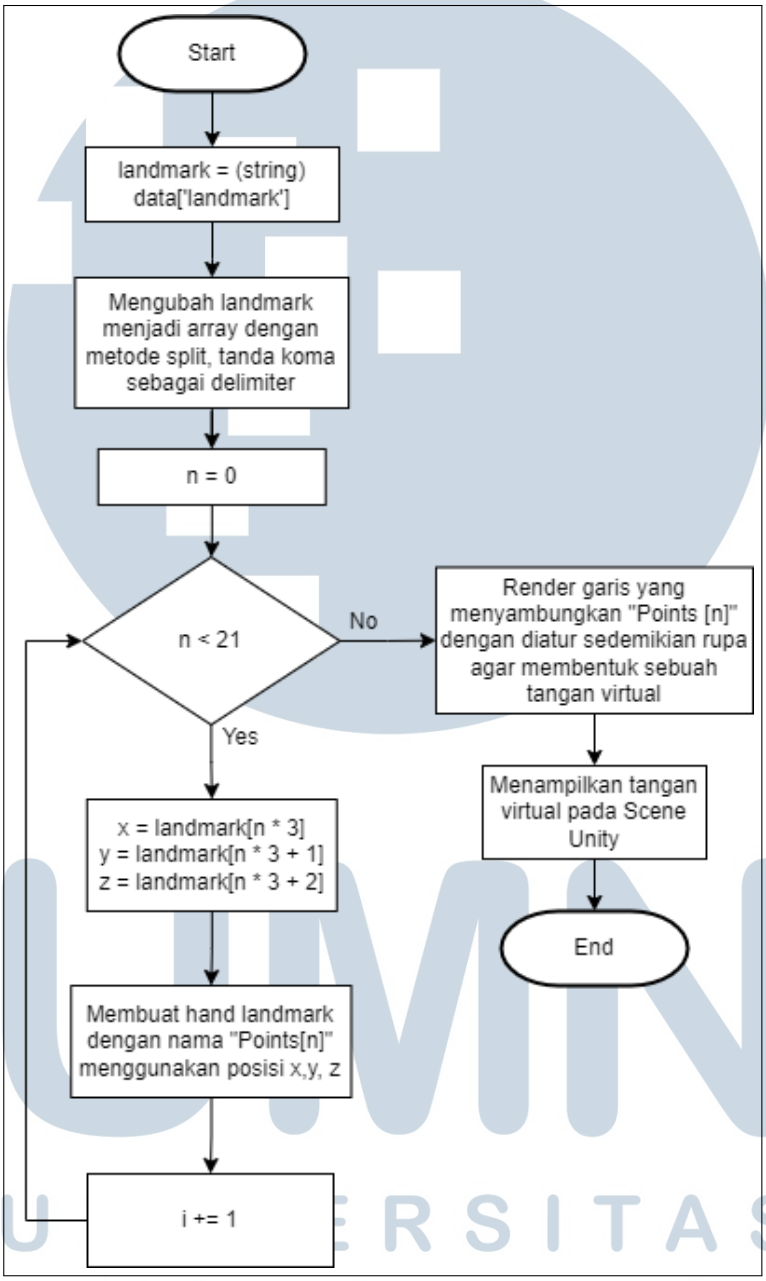


Gambar 3.6. *Flowchart* Utama *Hand Tracking Receiver* pada Unity

A Flowchart Build Virtual Hand

Gambar 3.7 merupakan *flowchart* dari *Build Virtual Hand*. Setelah data JSON diterima dan di-*decode* pada Unity, $data['landmark']$ dapat digunakan untuk membangun *virtual hand*. Tetapi karena data *landmark* yang diterima ini berbentuk *string*, diperlukan *pre-process* sederhana untuk mendapatkan koordinat yang diperlukan, yaitu X_n, Y_n, dan Z_n. Proses *split* pada $data['landmark']$ dilakukan untuk memisahkan data tersebut dengan menggunakan (,) atau tanda koma sebagai *delimiter*, proses ini menghasilkan array $data['landmark']$ dengan ukuran 63. Proses *looping* meng-interasi seluruh 21 koordinat *landmark* dan mengambil data X_{n*3}, Y_{n*3+1}, X_{n*3+2}, agar dapat mendapatkan koordinat dari tiap *landmark*. Kemudian, dibuat sebuah titik[n] atau *Points[n]* sesuai dengan 21 koordinat yang ada. Setelah semua titik dibuat, sebuah garis menyambungkan titik-titik tersebut sedemikian rupa agar dapat membentuk sebuah tangan. Referensi cara

menyambungkan titik tersebut dapat dilihat pada gambar 2.1, dimana garis hijau tersebut menandakan garis yang dihasilkan.

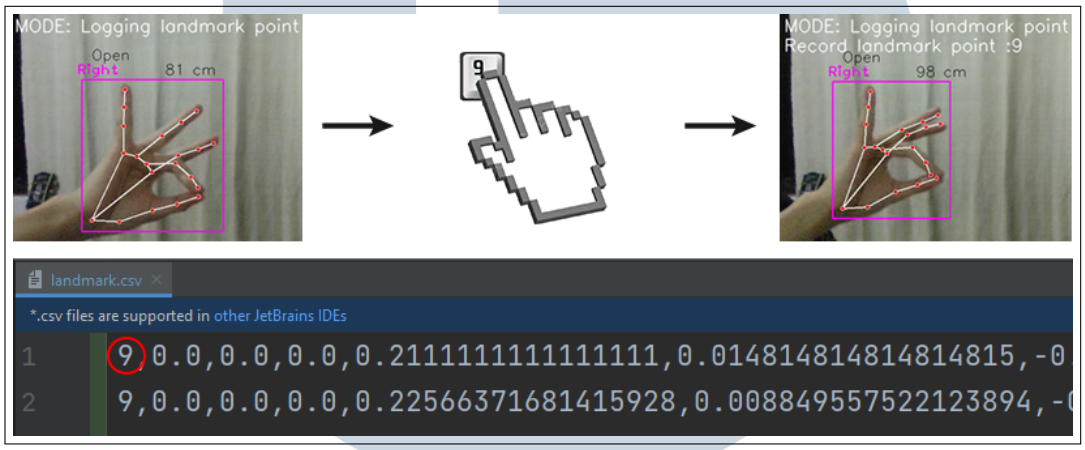


Gambar 3.7. *Flowchart Build Virtual Hand*

3.3 Pengumpulan Dataset

Dataset dikumpulkan melalui aplikasi Python yang telah dijelaskan pada bagian 3.2.1. Untuk mengumpulkan dataset, aplikasi Python tersebut

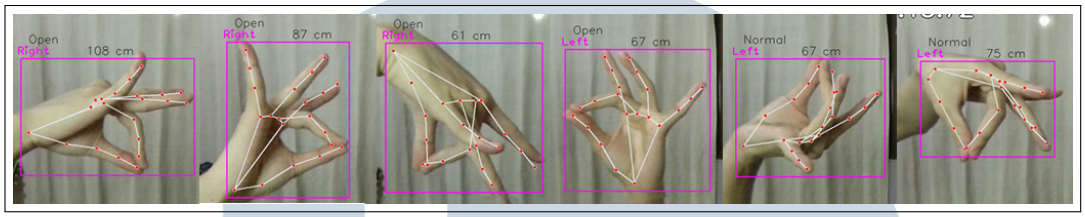
memiliki mode *Collect Data*, yang memiliki fungsi untuk merekam *dataset* dan menyimpannya ke dalam sebuah file CSV. Contoh cara kerja *Collect Data* dapat dilihat pada gambar 3.8. Pertama, tampilkan gesture yang ingin direkam pada aplikasi, jika sudah yakin, tekan pada *keyboard* angka mulai dari angka 0 - 9. Setelah itu, data *landmark* direkam di dalam sebuah file bernama landmark.csv. Perlu diperhatikan bahwa angka yang ditekan ditambahkan dipaling depan data *landmark*, angka ini menjadi *label* untuk data *gesture* tersebut.



Gambar 3.8. *Mode Collect Data* pada aplikasi Python

Proses *Collect Data* ini dapat dilakukan berulang-ulang dengan cara mengganti posisi tangan, tetapi dengan tetap menampilkan *gesture* yang sama untuk membuat dataset menjadi lebih bervariasi, seperti yang dapat dilihat pada gambar 3.9. Proses ini sebenarnya dapat digantikan dengan menggunakan sebuah *video*, tetapi metode manual dipilih karena data yang direkam dapat dikendalikan secara langsung dan dapat diperhatikan apakah data tersebut sesuai atau tidak. Berbeda apabila data direkam dengan menggunakan *video*, karena data yang terekam tidak bisa dikendalikan secara langsung. Jumlah sampel data yang dikumpulkan berkisar antara ∼300 baris untuk tiap *gesture*, yang terdiri atas ∼150 data tangan kanan, dan ∼150 data tangan kiri. Jumlah ini juga bervariasi, karena dapat disesuaikan dengan kebutuhan. Misalnya, apabila *gesture* yang ingin diklasifikasi hanya diperlukan pada tangan kiri, maka data yang dikumpulkan cukup berupa ∼150 data tangan kiri, dan sebaliknya. Dataset yang telah dikumpulkan memiliki jumlah total 3506 data, data ini terdiri dari 10 *gesture* yang dikumpulkan dengan cara yang telah dijelaskan diatas. Data yang dikumpulkan hanya untuk 6 *gesture* agar dapat mengurangi kompleksitas dari penelitian, sehingga penelitian ini dapat fokus untuk mengetahui seberapa efektif sebuah Dense Neural Network dalam melakukan klasifikasi untuk

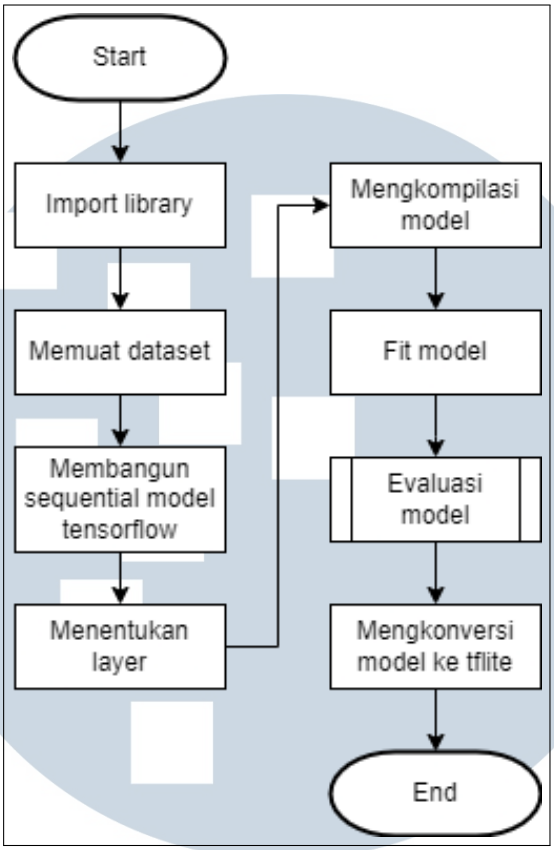
Hand Gesture, serta untuk mengetahui apakah mungkin *Hand Tracking* dan *Hand Gesture Classification* ini diimplementasikan ke dalam Unity [35].



Gambar 3.9. Contoh variasi dalam merekam *gesture*

3.4 Membuat Model

Gambar 3.10 merupakan gambar *flowchart* pembuatan model. Setelah mengimport semua *library* yang dibutuhkan, dataset yang sudah di-*upload* sebelumnya ke Google Drive di-*download* ke workspace Google Colab agar dapat digunakan. Setelah berhasil di-*download*, *dataset* diproses sederhana dengan cara di-*split* menjadi data X (data landmark) dan data Y (label). Kemudian menggunakan *library* tensorflow untuk membuat sebuah *sequential neural network model*. Model *sequential* ini mendistilisasi *input* data hingga mencapai *output* yang diinginkan. Kemudian layer dapat ditambahkan ke dalam *sequential model*. *Input layer*, *Dropout layer*, dan *Dense layer* ditambahkan dengan menggunakan aktivasi seperti ReLU untuk *hidden layer* dan Softmax pada *output layer*. Setelah semua layer ditentukan, model dapat di-compile. Setelah model dicompile, proses *training* dapat dimulai. Hasil *training* model dinilai dengan menggunakan metriks seperti *accuracy*, *precision*, *recall* dan *f1-score*. Terakhir, model dikonversi menjadi tflite agar dapat digunakan oleh *Interpreter* Tensorflow pada aplikasi di Python.



Gambar 3.10. *Flowchart* pembuatan model

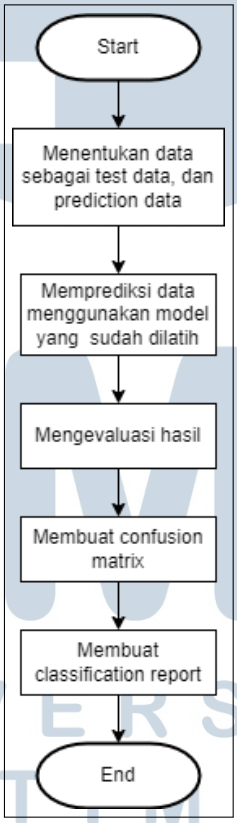
3.5 Implementasi

Setelah aplikasi Python untuk *Hand Tracking* dan model *Hand Gesture Classification* selesai dibuat, data JSON yang berisi kedua data tersebut dapat dikirim ke Unity melalui protokol UDP secara lokal. Unity menerima data dan mulai mengimplementasi data tersebut untuk membuat sebuah tangan virtual. Implementasi yang dibuat pada Unity memiliki konsep yang kurang lebih mirip seperti Virtual Reality (VR) Controller pada umumnya, dimana pengguna bisa menggunakan tangan virtual ini untuk berinteraksi dengan objek yang ada di Unity. Sebagai *demo*, dibuat sebuah *scene* pada Unity berisikan objek 3D yang dapat berinteraksi dengan tangan virtual pengguna. Agar dapat berinteraksi dengan objek, digunakan fungsi dari *Ray Tracing* dan *Collider* pada Unity, karena fungsi ini juga digunakan oleh VR Controller pada umumnya. Selain berinteraksi dengan menggunakan *Ray Tracing*, tangan virtual ini juga dapat berinteraksi dengan *User Interface (UI)* yang ditampilkan pada *scene* apabila pengguna menekan tombol tertentu. Interaksi ini juga mendemonstrasikan bagaimana sebuah *gesture* dapat

menambah kompleksitas bagi pengguna dalam berinteraksi dengan *environment 3D*. Misalnya dengan membuat suatu objek hanya bisa diakses / digunakan oleh pengguna dengan *gesture* tertentu.

3.6 Evaluasi

Gambar 3.11 menunjukkan *flowchart* dari proses Evaluasi model. Setelah model dilatih, evaluasi dilakukan dengan menggunakan *test data* dan *prediction data* untuk melakukan prediksi dengan menggunakan model yang sudah dilatih sebelumnya. Penulis juga membuat sebuah *confusion matrix* untuk mempermudah melihat performa klasifikasi secara visual. Setelah itu, metriks evaluasi *accuracy*, *precision*, *recall*, dan *f1-score* juga dibuat dengan menggunakan *classification report* dari *library sklearn*.



Gambar 3.11. *Flowchart* Evaluasi Model