BAB 3
METODOLOGI PENELITIAN

3.1 Pengumpulan Data

Pengumpulan data dalam penelitian ini dilakukan dengan mengumpulkan file dokumen kalibrasi yang digunakan oleh BSN dari berbagai laboratorium, seperti Laboratorium Kelistrikan, Pengukuran Panjang, dan Kalibrasi Suhu. Data utama yang dikumpulkan meliputi file Excel berisi hasil kalibrasi dan templat DCC dalam format Word. Selain itu, dilakukan wawancara dengan Ibu Hayati Amalia selaku Koordinator Tim Metrologi Digital BSN untuk memahami alur kerja manual dan tantangan yang dihadapi. Transkrip wawancara dapat diakses pada google drive.

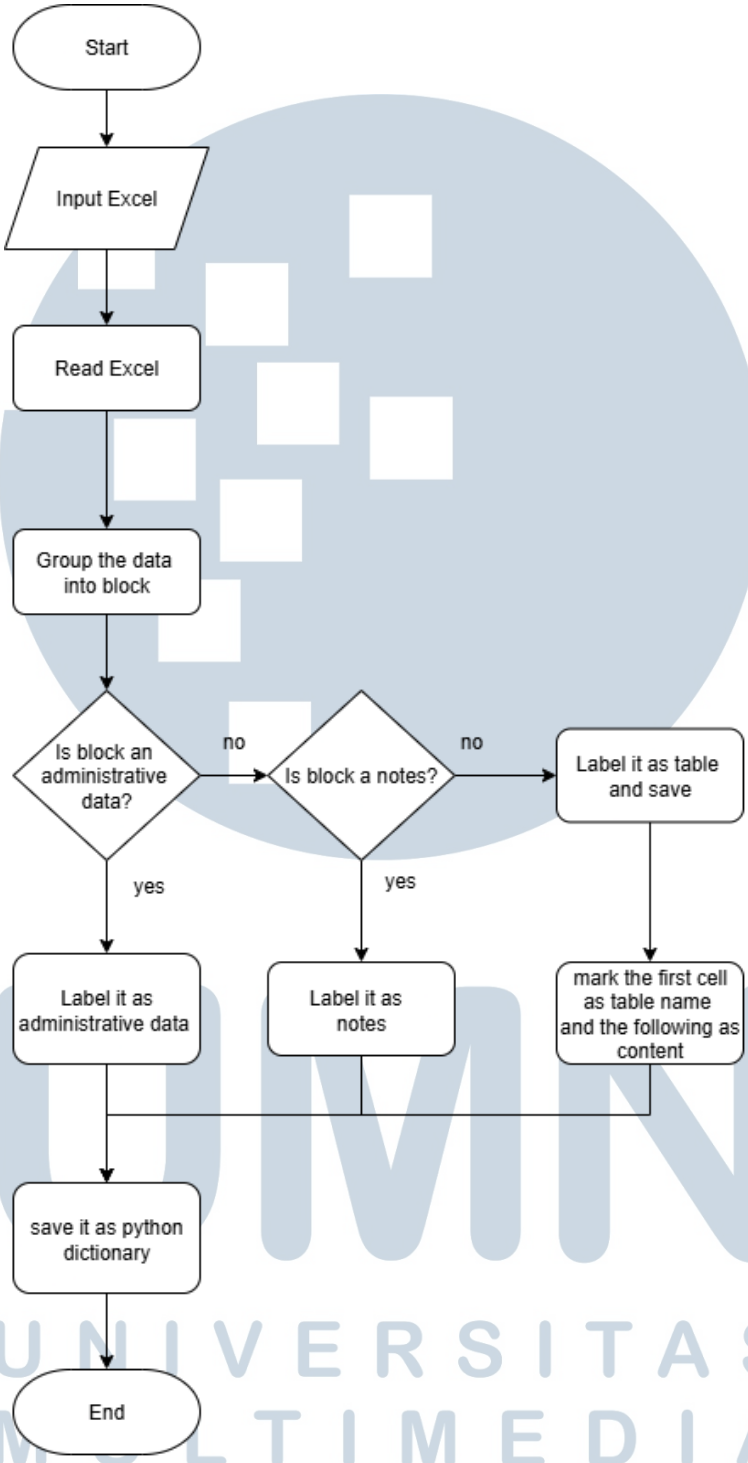
3.2 Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk menentukan spesifikasi sistem. Berdasarkan wawancara dan analisis dokumen, teridentifikasi bahwa tantangan utama adalah tingginya variasi format pada lebih dari 40 templat. Oleh karena itu, dirumuskan kebutuhan fungsional utama, yaitu sistem harus mampu secara cerdas:

- Mengenali dan mengolah berbagai format data, termasuk *form* (sederhana dan majemuk), tabel (dengan struktur dinamis), dan catatan.
- Mengisi dokumen templat Word secara otomatis dan akurat sesuai dengan data yang telah diekstraksi.

3.3 Perancangan Sistem dan Alur Kerja

Sistem dirancang dengan arsitektur yang terdiri dari dua fase utama yang sekuensial. Fase pertama berfokus pada analisis tata letak dan ekstraksi data dari file Excel. Fase kedua berfokus pada populasi atau pengisian data yang telah terstruktur tersebut ke dalam templat Word. Alur kerja untuk masing-masing fase dirancang secara terpisah alur kerja dari fase pertama dapat dilihat pada Gambar 3.1.



Gambar 3.1. Alur kerja Fase 1: Ekstraksi dan Kategorisasi Data dari Excel.

3.4 Fase 1: Ekstraksi dan Kategorisasi Data dari Excel

Tujuan dari fase ini adalah mengubah data semi-terstruktur dari Excel menjadi sebuah *dictionary* Python yang terorganisir. Proses ini dimulai dengan inisialisasi dan pemanggilan fungsi utama seperti pada Kode 3.1.

```
if __name__ == '__main__':
    excel_file = "path/to/your/excel_file.xlsx"
    word_template = "path/to/your/word_template.docx"
    output_file = "Laporan_Otomasi_Final.docx"

    print("--- Fase 1: Parsing Data Excel ---")
    data_laporan = parse_data_from_excel_stable(excel_file)
```

Kode 3.1: Inisialisasi dan pemanggilan fungsi utama.

3.4.1 Segmentasi Data menjadi Blok

Langkah pertama adalah memecah data di Excel menjadi blok-blok informasi logis (form, tabel, catatan) dengan menggunakan baris kosong sebagai pemisah. Hal ini dilakukan oleh fungsi `_group_rows_into_blocks` seperti yang ditunjukkan pada Kode 3.2. Fungsi ini membaca file Excel dan mengiterasi setiap baris; baris yang tidak kosong dikumpulkan, dan saat baris kosong ditemukan, kumpulan baris tersebut disimpan sebagai satu blok.

```
def _group_rows_into_blocks(excel_file_path, sheet_name='Lap'):
    df = pd.read_excel(excel_file_path, sheet_name=sheet_name,
    header=None)
    blocks, current_block = [], []
    for _, row in df.iterrows():
        is_empty_row = row.isnull().all()
        if is_empty_row:
            if current_block:
                blocks.append(current_block)
            current_block = []
        else:
            # Mengubah sel kosong (NaN) menjadi string ''
            row_list = [str(item) if pd.notna(item) else '' for
    item in row]
            current_block.append(row_list)
    if current_block:
        blocks.append(current_block)
```

```
    return blocks
```

Kode 3.2: Kode untuk segmentasi data Excel menjadi blok logis.

3.4.2 Klasifikasi dan Ekstraksi Blok

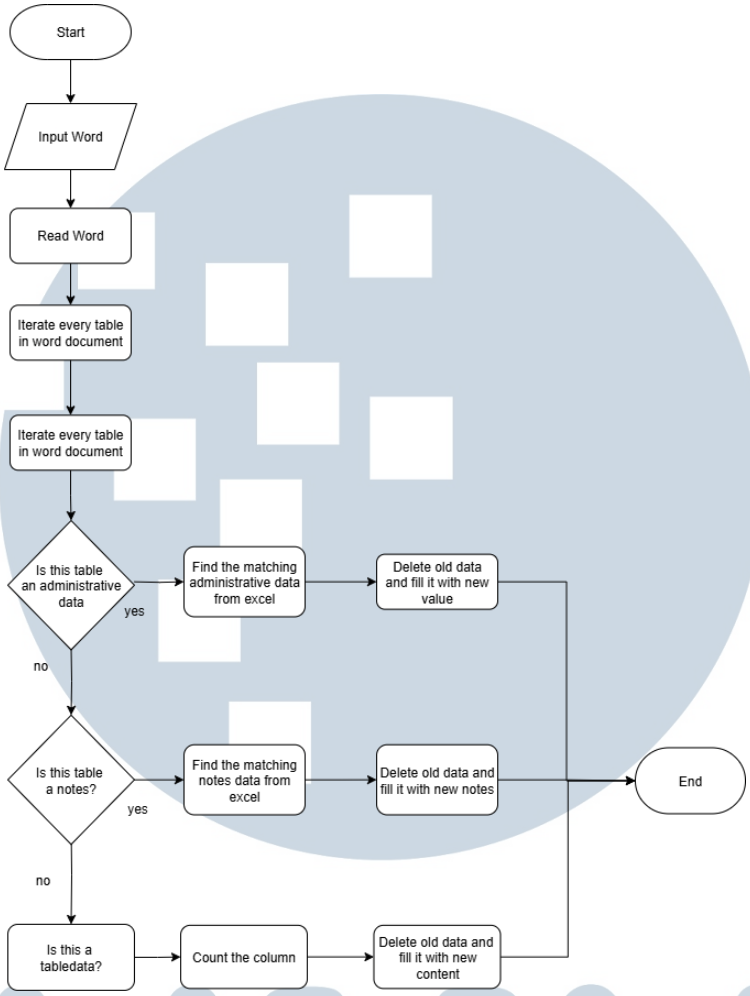
Setelah data terbagi menjadi blok, setiap blok dianalisis dan diklasifikasikan menggunakan aturan heuristik seperti pada Kode 3.3. Sistem akan mengklasifikasikan blok menjadi tiga kategori: blok ”Notes” diidentifikasi melalui kata kunci; blok ”Form” diidentifikasi dengan menghitung persentase baris yang berisi ’:’; dan blok lainnya dianggap sebagai ”Tabel Data”.

```
for block in blocks:
    first_line_str = ' '.join(block[0]).strip()
    if 'Catatan / Notes' in first_line_str:
        # Proses ekstraksi data catatan
        report_data['notes'].extend(...)
    elif _is_header_block(block):
        # Proses ekstraksi data form (key-value)
        for line in block:
            key, value = line.split(':', 1)
            report_data['header'][key.strip()] = value.strip()
    else:
        # Proses ekstraksi data tabel (header dan baris data)
        table_title = ' '.join(block[0])
        # ... (logika pemilahan header dan data rows)
        data_rows.append(line[1:]) # Membuang kolom kosong
```

Kode 3.3: Logika untuk klasifikasi dan ekstraksi setiap blok data.

3.5 Fase 2: Populasi Data ke Templat Word

Selanjutnya alur kerja untuk Fase 2, seperti yang diilustrasikan pada Gambar 3.2, berfokus pada proses populasi atau pengisian data yang telah terstruktur ke dalam templat Microsoft Word. Pada tahap ini, *dictionary* Python yang dihasilkan dari Fase 1 digunakan sebagai sumber data untuk mengisi dokumen templat secara otomatis. Proses ini melibatkan beberapa langkah kunci, yaitu memuat dokumen templat, melakukan iterasi pada setiap tabel, menerapkan logika untuk membedakan dan mengisi tabel *form* dan tabel data, serta memformat sel sebelum menyimpan dokumen akhir..



Gambar 3.2. Alur kerja Fase 2: Populasi Data ke Templat Word.

3.5.1 Input Template Word dan Iterasi Tabel di Dalam File Word

Sistem pertama-tama memuat dokumen Word menggunakan pustaka `python-docx`, lalu memulai iterasi pada setiap tabel yang ada di dalamnya, seperti pada Kode 3.4.

```
def generate_report_comeback(data, template_path, output_path):
    doc = Document(template_path)
    for table in doc.tables:
        # ... (logika penanganan tabel)
```

Kode 3.4: Kode untuk memuat templat dan iterasi tabel Word.

3.5.2 Pencocokan dan Populasi Data

Di dalam *loop*, setiap tabel dievaluasi. Kode 3.5 menunjukkan bagaimana sistem membedakan antara tabel form 2 kolom dan tabel data lebih dari data 2 kolom. Untuk tabel form, sistem mencocokkan kunci di sel pertama. Untuk tabel data, sistem mencocokkan judul tabel, lalu menghapus konten lama dan mengisi dengan data baru.

```
# Di dalam loop 'for table in doc.tables:'
if len(table.columns) == 2:
    # Logika untuk mengisi tabel form
    for row in table.rows:
        key_in_word = row.cells[0].text.strip()
        if key_in_word in data['header']:
            set_cell_text(row.cells[1], data['header'][key_in_word
    ])
else:
    # Logika untuk mengisi tabel data
    title_in_word = table.rows[0].cells[0].text.strip()
    # ... (logika pencocokan judul)
    update_table_headers(table, header_data)
    populate_table_data(table, data_rows)
```

Kode 3.5: Logika untuk pencocokan dan populasi data ke tabel Word.

3.5.3 Pemformatan dan Penyimpanan

Setiap kali sebuah sel diisi, fungsi `set_cell_text` dipanggil untuk memastikan format yang konsisten. Di akhir proses, dokumen disimpan sebagai file baru. Kode 3.6 menunjukkan kedua fungsi ini.

```
def set_cell_text(cell, text, bold=False, align='LEFT'):
    cell.text = ''
    p = cell.paragraphs[0]
    p.alignment = ...
    run = p.add_run(str(text))
    run.font.name = 'Arial'
    run.font.size = Pt(9)
    run.font.bold = bold

# Di akhir fungsi generate_report_comeback
doc.save(output_path)
```

Kode 3.6: Kode untuk pemformatan sel dan penyimpanan dokumen.