DAFTAR PUSTAKA

Al Kadafi, A.J., 2017. Deteksi Objek Penghalang Secara Real-Time Berbasis Mobile Bagi Penyandang Tunanetra Menggunakan Analisis Blob (Sarjana). Universitas Brawijaya.

Aningtiyas, Prisky Ratna, Agus Sumin, and Setia Wirawan. Pembuatan Aplikasi Deteksi Objek Menggunakan TensorFlow Object Detection API dengan Memanfaatkan SSD MobileNet V2 Sebagai Model Pra-Terlatih. *Jurnal Ilmiah Komputasi* 19.3 (2020): 421-430.

Aulia, Dimas, Rayhan Atala, Thufail Qolba. 2021. Buku Panduan Pembelajaran Python 3. Jakarta: Divisi Software Development.

Basuki, Lutif Febriandita. 2016. Implementasi Metode Histograms of Oriented Gradients dengan Optimasi Algoritma Frei-Chen untuk Deteksi Citra Manusia. http://elib.unikom.ac.id/gdl.php?mod=browse&op=read&id=jbptunikomp p-gdl-lutfifebri-35958 (diakses 7 Februari 2022).

Dewi Poerwanti, S., 2017. Pengelolaan Tenaga Kerja Difabel untuk Mewujudkan Workplace Inclusion. INKLUSI 4, 1.

Gianani, S., Mehta, A., Motwani, T., Shende, R., 2018. JUVO - An Aid for the Visually Impaired, in: 2018 International Conference on Smart City and Emerging Technology (ICSCET). Presented at the 2018 International Conference on Smart City and Emerging Technology (ICSCET), pp. 1–4.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Murphy, K. 2017. Speed/accuracy trade-offs for modern convolutional object detectors. *Proceedings – 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017–Janua*, 3296–3305.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., C. Berg, A., 2016. SSD: Single Shot MultiBox Detector, in: Computer Vision – ECCV 2016. Presented at the Computer Vision – ECCV 2016, pp. 21–37.

Majumder, A., Gopi, M., 2018. Introduction to Visual Computing - Core Concepts in Computer Vision, Graphics, and Image Processing. CRC Press, London, New York.

Mane, S.S., Yangandul, C.G., 2016. Calculating the dimensions of an object using a single camera by learning the environAment, in: 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (ICATCCT), pp 457-460.

Parihar, A.S., Gupta, M., Sikka, V., Kaur, G., 2017. Dimensional analysis of objects in a 2D image, in: 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT). Presented at the 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1–7.

Pertuni (Persatuan Tunanetra Indonesia). 2022. Terwujudnya masyarakat inklusif. https://pertuni.or.id (Diakses 17 Januari 2022).

Prabowo, Dedy Agung, and Dedy Abdullah. 2018. Deteksi dan perhitungan objek berdasarkan warna menggunakan Color Object Tracking. *Pseudocode* 5.2 (2018): 85-91.

Rahardja, Djaja. 2009 "Konsep dan Strategi Implementasi KTSP SLB Tunanetra". Pendidikan Luar Biasa, Bandung.

Rahman, S., Ullah, Sana, Ullah, Sehat, 2018. Obstacle Detection in Indoor Environment for Visually Impaired Using Mobile Camera. In *Journal of Physics: Conference Series* Vol. 960, No. 1, p. 012046.

Ranalli, L., Di Stefano, L., Plebani, E., Falchetto, M., Pau, D., D'Alto, V., 2018. Automated Generation of a Single Shot Detector C Library from High Level Deep Learning *Frame*works, in: 2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI). Presented at the 2018 IEEE 4th International Forum on Research and Technology for

Society and Industry (RTSI), IEEE, Palermo, pp. 1–4. https://doi.org/10.1109/RTSI.2018.8548427

Ryu, J., Kim, S., 2018. Chinese character detection using modified Single Shot Multibox Detector, in: 2018 18th International Conference on Control, Automation and Systems (ICCAS). Presented at the 2018 18th International Conference on Control, Automation and Systems (ICCAS), pp. 1313–1315.

Supriyadi, Agung., 2018. 5 Pengendalian bahaya naik dan turun tangga. https://katigaku.top/2018/07/13/5-pengendalian-bahaya-naik-dan-turun-tangga/. (diakses 15 Januari 2022).

Syawaluddin, 2016. Pengenalan Plat Nomor Otomatis Menggunakan Principal Component Analysis (PCA) dan Learning Vector Quantization (LVQ). Teknik Informatika, Telkom University.

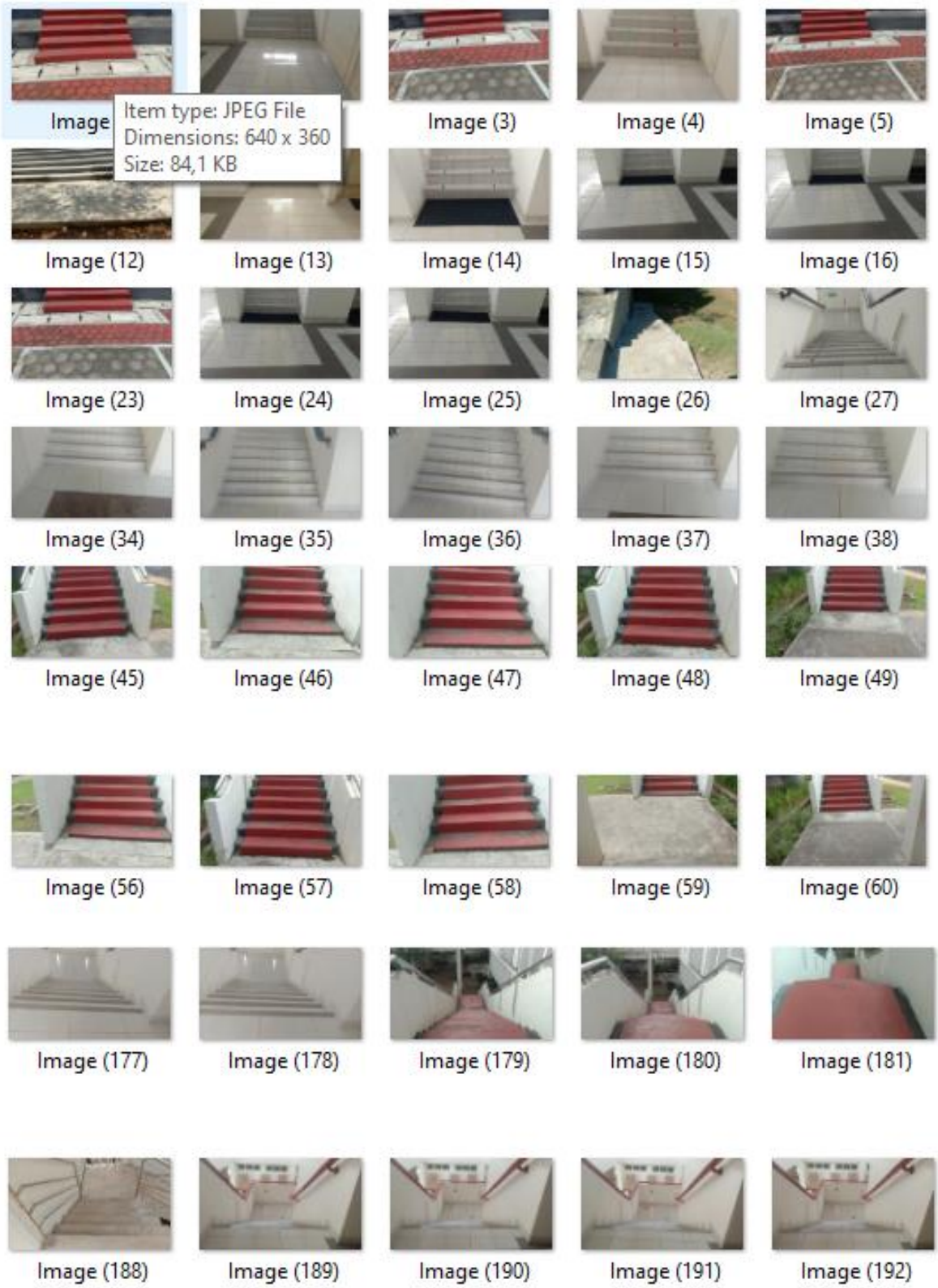
Sutoyo, T, Edy Mulyanto, Vincent Suhartono, Oky Nurhayati, dan Wijayanarto. 2009. Teori Pengolahan Citra Digital. Yogyakarta : Andi Offset.

Utami, Ema dan Suwanto Raharjo. 2004. Logika, Algoritma dan Implementasinya dalam Bahasa Python di GNU/Linux. Yogyakarta : Andi Offset.

WHO. 2010. Global data visual impairments 2010. http://www.who.int/ GLOBALDATAFINALforweb.pdf. (Diakses 18 Januari 2022).

LAMPIRAN

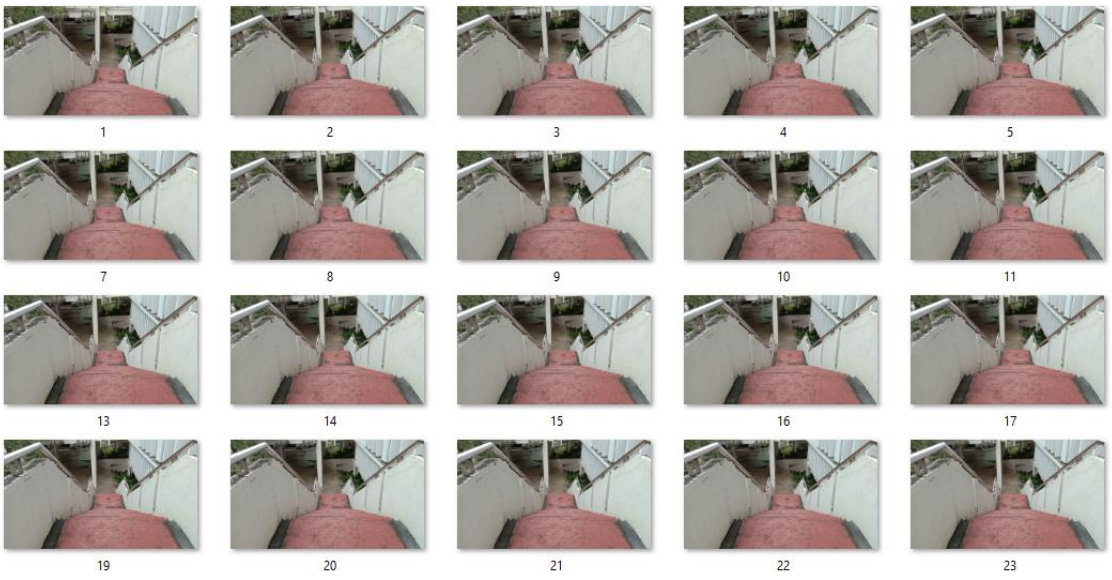
1. Contoh Data Training



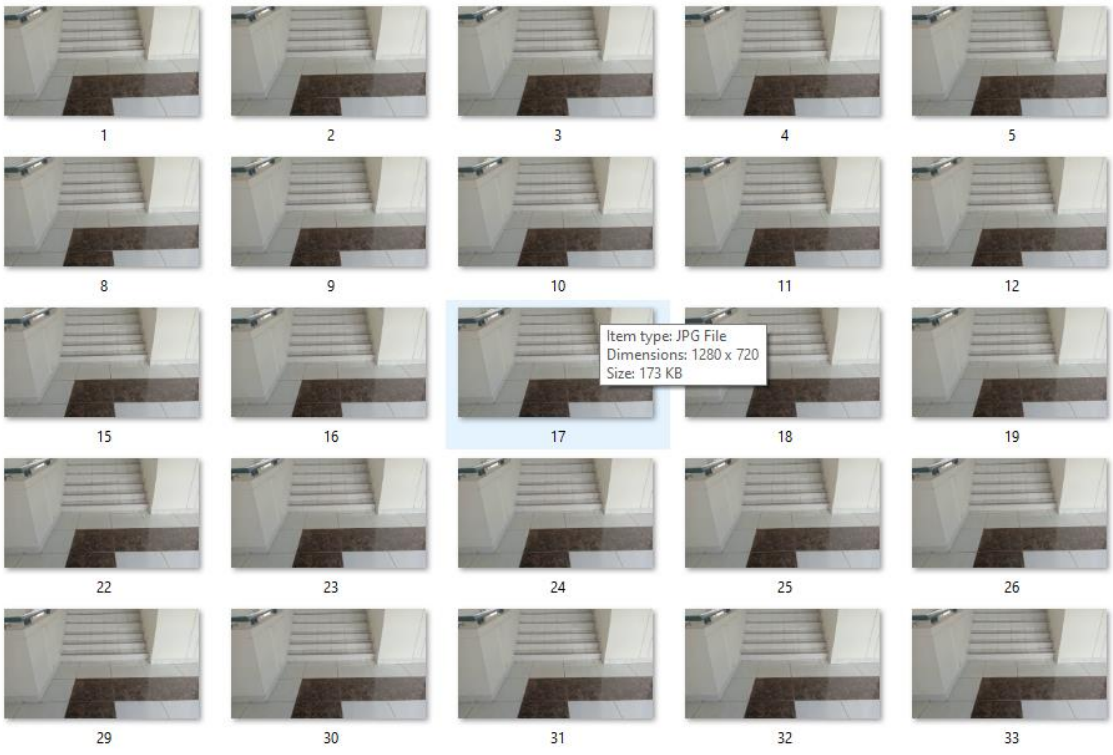
2. Contoh Data Testing

- **Contoh *Frame* Tangga Naik Gedung CSA**

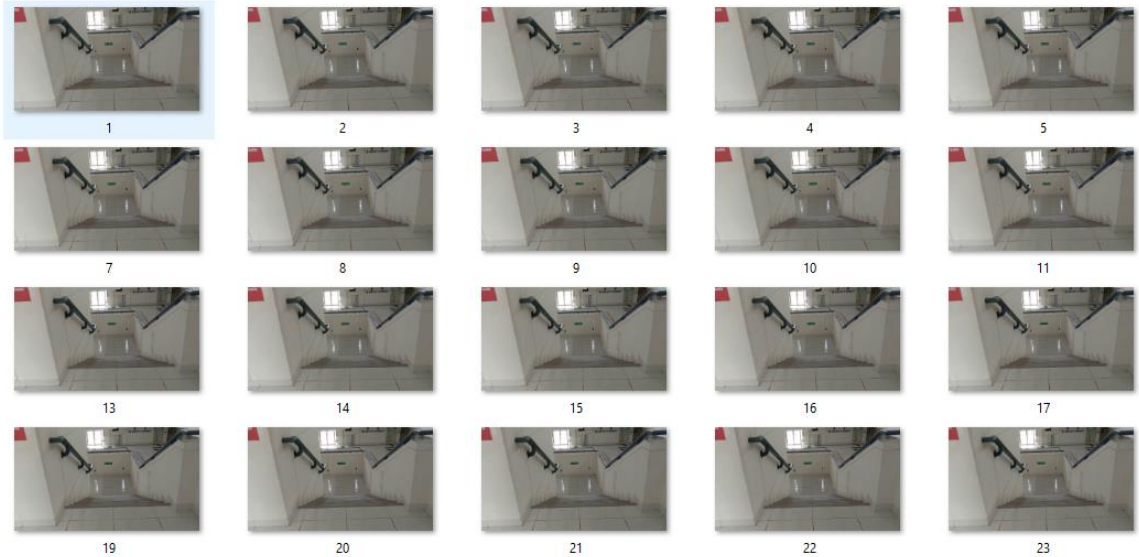
- **Contoh *Frame* Tangga Turun Gedung CSA**



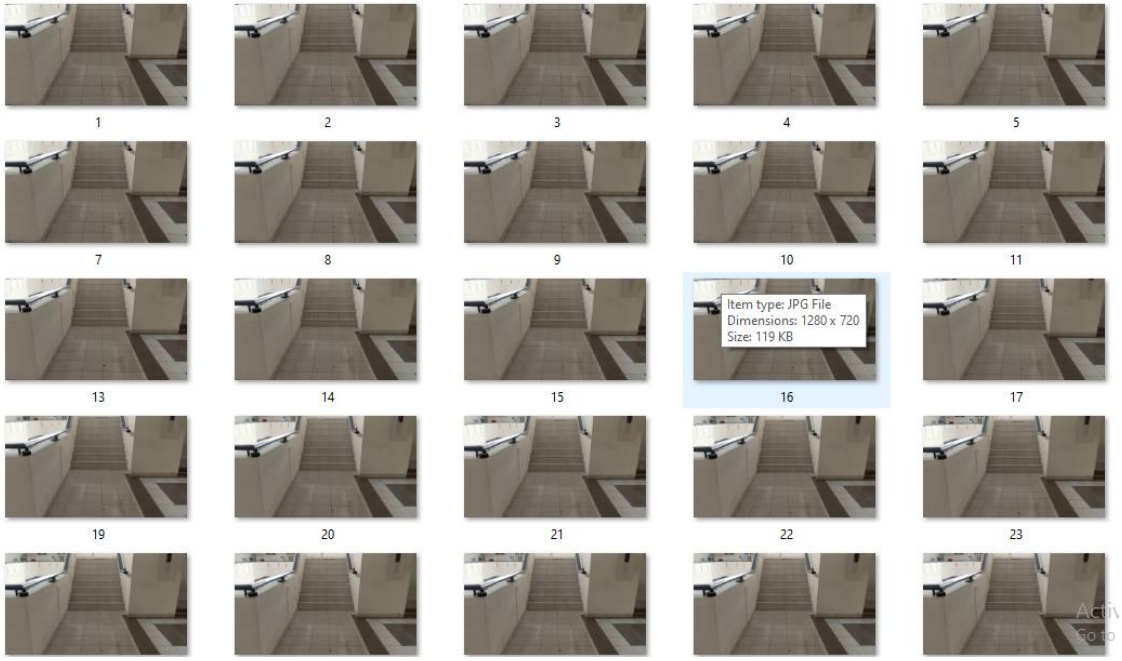
- **Contoh *Frame* Tangga Naik Gedung Elektro**



- **Contoh *Frame* Tangga Turun Gedung Elektro**

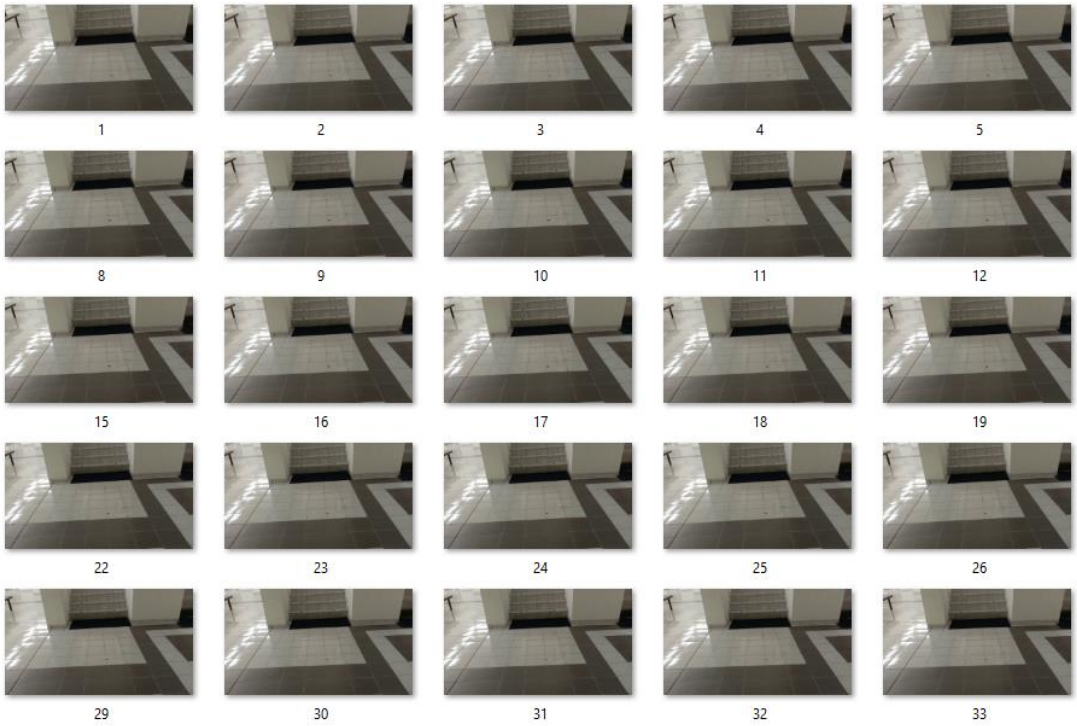


- **Contoh *Frame* Tangga Naik Gedung Sipil**



- **Contoh *Frame* Tangga Turun Gedung Sipil**

- **Contoh *Frame* Tangga Naik Gedung Geologi**



- **Contoh *Frame* Tangga Turun Gedung Geologi**

3. Training Model SSD

```
import json
import os
import tensorflow as tf

from object_detection.builders import dataset_builder
from object_detection.builders import graph_rewriter_builder
from object_detection.builders import model_builder
from object_detection.legacy import trainer
from object_detection.utils import config_util

tf.logging.set_verbosity(tf.logging.INFO)

flags = tf.app.flags
flags.DEFINE_string('master', '', 'Name of the TensorFlow master to use.')
flags.DEFINE_integer('task', 0, 'task id')
flags.DEFINE_integer('num_clones', 1, 'Number of clones to deploy per
worker.')
flags.DEFINE_boolean('clone_on_cpu', False,
                     'Force clones to be deployed on CPU.  Note that even if '
                     'set to False (allowing ops to run on gpu), some ops may '
                     'still be run on the CPU if they have no GPU kernel.')
flags.DEFINE_integer('worker_replicas', 1, 'Number of worker+trainer '
                     'replicas.')
flags.DEFINE_integer('ps_tasks', 0,
                     'Number of parameter server tasks. If None, does not use '
                     'a parameter server.')
flags.DEFINE_string('train_dir', '',
                    'Directory to save the checkpoints and training summaries.')

flags.DEFINE_string('pipeline_config_path', '',
                    'Path to a pipeline_pb2.TrainEvalPipelineConfig config '
                    'file. If provided, other configs are ignored')

flags.DEFINE_string('train_config_path', '',
                    'Path to a train_pb2.TrainConfig config file.')
flags.DEFINE_string('input_config_path', '',
                    'Path to an input_reader_pb2.InputReader config file.')
flags.DEFINE_string('model_config_path', '',
```

```
                'Path to a model_pb2.DetectionModel config file.')

FLAGS = flags.FLAGS


@tf.contrib.framework.deprecated(None, 'Use
object_detection/model_main.py.')
def main(_):
  assert FLAGS.train_dir, '`train_dir` is missing.'
  if FLAGS.task == 0: tf.gfile.MakeDirs(FLAGS.train_dir)
  if FLAGS.pipeline_config_path:
    configs = config_util.get_configs_from_pipeline_file(
        FLAGS.pipeline_config_path)
    if FLAGS.task == 0:
      tf.gfile.Copy(FLAGS.pipeline_config_path,
                    os.path.join(FLAGS.train_dir, 'pipeline.config'),
                    overwrite=True)
  else:
    configs = config_util.get_configs_from_multiple_files(
        model_config_path=FLAGS.model_config_path,
        train_config_path=FLAGS.train_config_path,
        train_input_config_path=FLAGS.input_config_path)
    if FLAGS.task == 0:
      for name, config in [('model.config', FLAGS.model_config_path),
                           ('train.config', FLAGS.train_config_path),
                           ('input.config', FLAGS.input_config_path)]:
        tf.gfile.Copy(config, os.path.join(FLAGS.train_dir, name),
                      overwrite=True)

  model_config = configs['model']
  train_config = configs['train_config']
  input_config = configs['train_input_config']

  model_fn = functools.partial(
      model_builder.build,
      model_config=model_config,
      is_training=True)

  def get_next(config):
    return dataset_builder.make_initializable_iterator(
```

```
    dataset_builder.build(config)).get_next()

create_input_dict_fn = functools.partial(get_next, input_config)

env = json.loads(os.environ.get('TF_CONFIG', '{}'))
cluster_data = env.get('cluster', None)
cluster = tf.train.ClusterSpec(cluster_data) if cluster_data else None
task_data = env.get('task', None) or {'type': 'master', 'index': 0}
task_info = type('TaskSpec', (object,), task_data)

# Parameters for a single worker.
ps_tasks = 0
worker_replicas = 1
worker_job_name = 'lonely_worker'
task = 0
is_chief = True
master = ''

if cluster_data and 'worker' in cluster_data:
  # Number of total worker replicas include "worker"s and the "master".
  worker_replicas = len(cluster_data['worker']) + 1
if cluster_data and 'ps' in cluster_data:
  ps_tasks = len(cluster_data['ps'])

if worker_replicas > 1 and ps_tasks < 1:
  raise ValueError('At least 1 ps task is needed for distributed training.')

if worker_replicas >= 1 and ps_tasks > 0:
  # Set up distributed training.
  server = tf.train.Server(tf.train.ClusterSpec(cluster), protocol='grpc',
                           job_name=task_info.type,
                           task_index=task_info.index)
  if task_info.type == 'ps':
    server.join()
    return

  worker_job_name = '%s/task:%d' % (task_info.type, task_info.index)
  task = task_info.index
  is_chief = (task_info.type == 'master')
  master = server.target
```

```
graph_rewriter_fn = None
if 'graph_rewriter_config' in configs:
  graph_rewriter_fn = graph_rewriter_builder.build(
      configs['graph_rewriter_config'], is_training=True)

trainer.train(
    create_input_dict_fn,
    model_fn,
    train_config,
    master,
    task,
    FLAGS.num_clones,
    worker_replicas,
    FLAGS.clone_on_cpu,
    ps_tasks,
    worker_job_name,
    is_chief,
    FLAGS.train_dir,
    graph_hook_fn=graph_rewriter_fn)


if __name__ == '__main__':
  tf.app.run()
```

4. Export Graph Model

```
import tensorflow as tf
from google.protobuf import text_format
from object_detection import exporter
from object_detection.protos import pipeline_pb2

slim = tf.contrib.slim
flags = tf.app.flags

flags.DEFINE_string('input_type', 'image_tensor', 'Type of input node.
Can be '
                    'one of [`image_tensor`, `encoded_image_string_tensor`, '
                    '`tf_example`]')
```

```
flags.DEFINE_string('input_shape', None,
                    'If input_type is `image_tensor`, this can explicitly set '
                    'the shape of this input tensor to a fixed size. The '
                    'dimensions are to be provided as a comma-separated list '
                    'of integers. A value of -1 can be used for unknown '
                    'dimensions. If not specified, for an `image_tensor, the '
                    'default shape will be partially specified as '
                    '`[None, None, None, 3]`.')
flags.DEFINE_string('pipeline_config_path', None,
                    'Path to a pipeline_pb2.TrainEvalPipelineConfig config '
                    'file.')
flags.DEFINE_string('trained_checkpoint_prefix', None,
                    'Path to trained checkpoint, typically of the form '
                    'path/to/model.ckpt')
flags.DEFINE_string('output_directory', None, 'Path to write outputs.')
flags.DEFINE_string('config_override', '',
                    'pipeline_pb2.TrainEvalPipelineConfig '
                    'text proto to override pipeline_config_path.')
flags.DEFINE_boolean('write_inference_graph', False,
                     'If true, writes inference graph to disk.')
tf.app.flags.mark_flag_as_required('pipeline_config_path')
tf.app.flags.mark_flag_as_required('trained_checkpoint_prefix')
tf.app.flags.mark_flag_as_required('output_directory')
FLAGS = flags.FLAGS


def main(_):
  pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
  with tf.gfile.GFile(FLAGS.pipeline_config_path, 'r') as f:
    text_format.Merge(f.read(), pipeline_config)
  text_format.Merge(FLAGS.config_override, pipeline_config)
  if FLAGS.input_shape:
    input_shape = [
        int(dim) if dim != '-1' else None
        for dim in FLAGS.input_shape.split(',')
    ]
  else:
    input_shape = None
  exporter.export_inference_graph(
```

FLAGS.input_type, pipeline_config,
FLAGS.trained_checkpoint_prefix,
FLAGS.output_directory, input_shape=input_shape,
write_inference_graph=FLAGS.write_inference_graph)

if __name__ == '__main__':
tf.app.run()

5. Deteksi Tiang

```
public class TensorFlowObjectDetectionAPIModel
implements Classifier {
  private static final Logger LOGGER = new Logger();

  // Only return this many results.
  private static final int MAX_RESULTS = 100;

  // Config values.
  private String inputName;
  private int inputSize;

  // Pre-allocated buffers.
  private Vector<String> labels = new
Vector<String>();
  private int[] intValues;
  private byte[] byteValues;
  private float[] outputLocations;
  private float[] outputScores;
  private float[] outputClasses;
  private float[] outputNumDetections;
  private String[] outputNames;

  private boolean logStats = false;

  private TensorFlowInferenceInterface
inferenceInterface;

  /**
   * Initializes a native TensorFlow session for
```

```
classifying images.
   *
   * @param assetManager The asset manager to be
used to load assets.
   * @param modelFilename The filepath of the model
GraphDef protocol buffer.
   * @param labelFilename The filepath of label file
for classes.
   */
  public static Classifier create(
      final AssetManager assetManager,
      final String modelFilename,
      final String labelFilename,
      final int inputSize) throws IOException {
    final TensorFlowObjectDetectionAPIModel d = new
TensorFlowObjectDetectionAPIModel();

    InputStream labelsInput = null;
    String actualFilename =
labelFilename.split("file:///android_asset/")[1];
    labelsInput = assetManager.open(actualFilename);
    BufferedReader br = null;
    br = new BufferedReader(new
InputStreamReader(labelsInput));
    String line;
    while ((line = br.readLine()) != null) {
      LOGGER.w(line);
      d.labels.add(line);
    }
    br.close();


    d.inferenceInterface = new
TensorFlowInferenceInterface(assetManager,
modelFilename);

    final Graph g = d.inferenceInterface.graph();

    d.inputName = "image_tensor";
    // The inputName node has a shape of [N, H, W,
C], where
```

```
    // N is the batch size
    // H = W are the height and width
    // C is the number of channels (3 for our
purposes - RGB)
    final Operation inputOp =
g.operation(d.inputName);
    if (inputOp == null) {
      throw new RuntimeException("Failed to find
input Node '" + d.inputName + "'");
    }
    d.inputSize = inputSize;
    // The outputScoresName node has a shape of [N,
NumLocations], where N
    // is the batch size.
    final Operation outputOp1 =
g.operation("detection_scores");
    if (outputOp1 == null) {
      throw new RuntimeException("Failed to find
output Node 'detection_scores'");
    }
    final Operation outputOp2 =
g.operation("detection_boxes");
    if (outputOp2 == null) {
      throw new RuntimeException("Failed to find
output Node 'detection_boxes'");
    }
    final Operation outputOp3 =
g.operation("detection_classes");
    if (outputOp3 == null) {
      throw new RuntimeException("Failed to find
output Node 'detection_classes'");
    }

    // Pre-allocate buffers.
    d.outputNames = new String[] {"detection_boxes",
"detection_scores",

"detection_classes", "num_detections"};
    d.intValues = new int[d.inputSize *
d.inputSize];
    d.byteValues = new byte[d.inputSize *
```

```
d.inputSize * 3];
    d.outputScores = new float[MAX_RESULTS];
    d.outputLocations = new float[MAX_RESULTS * 4];
    d.outputClasses = new float[MAX_RESULTS];
    d.outputNumDetections = new float[1];
    return d;
  }

  private TensorFlowObjectDetectionAPIModel() {}

  @Override
  public List<Recognition> recognizeImage(final
Bitmap bitmap) {
    // Log this method so that it can be analyzed
with systrace.
    Trace.beginSection("recognizeImage");

    Trace.beginSection("preprocessBitmap");
    // Preprocess the image data to extract R, G and
B bytes from int of form 0x00RRGGBB
    // on the provided parameters.
    bitmap.getPixels(intValues, 0,
bitmap.getWidth(), 0, 0, bitmap.getWidth(),
bitmap.getHeight());

    for (int i = 0; i < intValues.length; ++i) {
      byteValues[i * 3 + 2] = (byte) (intValues[i] &
0xFF);
      byteValues[i * 3 + 1] = (byte) ((byteValues[i
* 3 + 0] = (byte) ((intValues[i] >> 16) & 0xFF);
    }
    Trace.endSection(); // preprocessBitmap

    // Copy the input data into TensorFlow.
    Trace.beginSection("feed");
    inferenceInterface.feed(intValues[i] >> 8) &
0xFF);
      inputName, byteValues, 1, inputSize,
inputSize, 3);
    Trace.endSection();
```

```
    // Run the inference call.
    Trace.beginSection("run");
    inferenceInterface.run(outputNames, logStats);
    Trace.endSection();

    // Copy the output Tensor back into the output
array.
    Trace.beginSection("fetch");
    outputLocations = new float[MAX_RESULTS * 4];
    outputScores = new float[MAX_RESULTS];
    outputClasses = new float[MAX_RESULTS];
    outputNumDetections = new float[1];
    inferenceInterface.fetch(outputNames[0],
outputLocations);
    inferenceInterface.fetch(outputNames[1],
outputScores);
    inferenceInterface.fetch(outputNames[2],
outputClasses);
    inferenceInterface.fetch(outputNames[3],
outputNumDetections);
    Trace.endSection();

    // Find the best detections.
    final PriorityQueue<Recognition> pq =
        new PriorityQueue<Recognition>(
            1,
            new Comparator<Recognition>() {
              @Override
              public int compare(final Recognition
lhs, final Recognition rhs) {
                // Intentionally reversed to put
high confidence at the head of the queue.
                return
Float.compare(rhs.getConfidence(),
lhs.getConfidence());
              }
            });

    // Scale them back to the input size.
    for (int i = 0; i < outputScores.length; ++i) {
      final RectF detection =
```

```
            new RectF(
                outputLocations[4 * i + 1] *
inputSize,
                outputLocations[4 * i] * inputSize,
                outputLocations[4 * i + 3] *
inputSize,
                outputLocations[4 * i + 2] *
inputSize);
        pq.add(
            new Recognition("" + i, labels.get((int)
outputClasses[i]), outputScores[i], detection));
      }

      final ArrayList<Recognition> recognitions = new
ArrayList<Recognition>();
      for (int i = 0; i < Math.min(pq.size(),
MAX_RESULTS); ++i) {
        recognitions.add(pq.poll());
      }
      Trace.endSection(); // "recognizeImage"
      return recognitions;
    }

    @Override
    public void enableStatLogging(final boolean
logStats) {
      this.logStats = logStats;
    }

    @Override
    public String getStatString() {
      return inferenceInterface.getStatString();
    }

    @Override
    public void close() {
      inferenceInterface.close();
    }
}
```

6. Estimasi Jarak

```
public class MultiBoxTracker {
    public String Hasil;
  private final Logger logger = new Logger();

  private static final float TEXT_SIZE_DIP = 18;

  // Maximum percentage of a box that can be
overlapped by another box at detection time.
Otherwise
  // the lower scored box (new or old) will be
removed.
  private static final float MAX_OVERLAP = 0.2f;

  private static final float MIN_SIZE = 16.0f;

  // Allow replacement of the tracked box with new
results if
  // correlation has dropped below this level.
  private static final float MARGINAL_CORRELATION
= 0.75f;

  // Consider object to be lost if correlation
falls below this threshold.
  private static final float MIN_CORRELATION =
0.3f;

  private static final int[] COLORS = {
          Color.BLUE, Color.RED, Color.GREEN,
Color.YELLOW, Color.CYAN, Color.MAGENTA,
Color.WHITE,
          Color.parseColor("#55FF55"),
Color.parseColor("#FFA500"),
Color.parseColor("#FF8888"),
          Color.parseColor("#AAAAFF"),
Color.parseColor("#FFFFAA"),
Color.parseColor("#55AAAA"),
          Color.parseColor("#AA33AA"),
Color.parseColor("#0D0068")
  };

  private final Queue<Integer> availableColors =
new LinkedList<Integer>();

  public ObjectTracker objectTracker;
```

```
  final List<Pair<Float, RectF>> screenRects = new
LinkedList<Pair<Float, RectF>>();

  private static class TrackedRecognition {
    ObjectTracker.TrackedObject trackedObject;
    RectF location;
    float detectionConfidence;
    int color;
    String title;
  }

  private final List<TrackedRecognition>
trackedObjects = new
LinkedList<TrackedRecognition>();

  private final Paint boxPaint = new Paint();

  private final float textSizePx;
  private final BorderedText borderedText;

  private Matrix frameToCanvasMatrix;

  private int frameWidth;
  private int frameHeight;

  private int sensorOrientation;
  private Context context;

  public MultiBoxTracker(final Context context) {
    this.context = context;
    for (final int color : COLORS) {
      availableColors.add(color);
    }

    boxPaint.setColor(Color.RED);
    boxPaint.setStyle(Style.STROKE);
    boxPaint.setStrokeWidth(12.0f);
    boxPaint.setStrokeCap(Cap.ROUND);
    boxPaint.setStrokeJoin(Join.ROUND);
    boxPaint.setStrokeMiter(100);

    textSizePx =
            TypedValue.applyDimension(
                    TypedValue.COMPLEX_UNIT_DIP,
TEXT_SIZE_DIP,
context.getResources().getDisplayMetrics());
```

```
    borderedText = new BorderedText(textSizePx);
  }

  private Matrix getFrameToCanvasMatrix() {
    return frameToCanvasMatrix;
  }

  public synchronized void drawDebug(final Canvas
canvas) {
    final Paint textPaint = new Paint();
    textPaint.setColor(Color.WHITE);
    textPaint.setTextSize(60.0f);

    final Paint boxPaint = new Paint();
    boxPaint.setColor(Color.RED);
    boxPaint.setAlpha(200);
    boxPaint.setStyle(Style.STROKE);

    for (final Pair<Float, RectF> detection :
screenRects) {
      final RectF rect = detection.second;
      canvas.drawRect(rect, boxPaint);
      canvas.drawText("" + detection.first,
rect.left, rect.top, textPaint);
      borderedText.drawText(canvas,
rect.centerX(), rect.centerY(), "" +
detection.first);
    }

    if (objectTracker == null) {
      return;
    }

    // Draw correlations.
    for (final TrackedRecognition recognition :
trackedObjects) {
      final ObjectTracker.TrackedObject
trackedObject = recognition.trackedObject;

      final RectF trackedPos =
trackedObject.getTrackedPositionInPreviewFrame();

      if
(getFrameToCanvasMatrix().mapRect(trackedPos)) {
        final String labelString =
String.format("%.2f",
trackedObject.getCurrentCorrelation());
```

```
        borderedText.drawText(canvas,
trackedPos.right, trackedPos.bottom, labelString);
      }
    }

    final Matrix matrix =
getFrameToCanvasMatrix();
    objectTracker.drawDebug(canvas, matrix);
  }

  public synchronized void trackResults(
          final List<Recognition> results, final
byte[] frame, final long timestamp) {
    logger.i("Processing %d results from %d",
results.size(), timestamp);
    processResults(timestamp, results, frame);
  }

  public synchronized void draw(final Canvas
canvas) {
    final boolean rotated = sensorOrientation %
180 == 90;
    final float multiplier =
            Math.min(canvas.getHeight() / (float)
(rotated ? frameWidth : frameHeight),
                    canvas.getWidth() / (float)
(rotated ? frameHeight : frameWidth));
    frameToCanvasMatrix =
            ImageUtils.getTransformationMatrix(
                    frameWidth,
                    frameHeight,
                    (int) (multiplier * (rotated ?
frameHeight : frameWidth)),
                    (int) (multiplier * (rotated ?
frameWidth : frameHeight)),
                    sensorOrientation,
                    false);
    for (final TrackedRecognition recognition :
trackedObjects) {
      final RectF trackedPos =
              (objectTracker != null)
                      ?
recognition.trackedObject.getTrackedPositionInPrev
iewFrame()
                      : new
RectF(recognition.location);
```

```
getFrameToCanvasMatrix().mapRect(trackedPos);
      boxPaint.setColor(recognition.color);
      boxPaint.setStrokeWidth(2.0f);

      final Paint paint = new Paint();
      paint.setColor(Color.YELLOW);
      paint.setStyle(Style.STROKE);
      paint.setStrokeWidth(2.0f);
      canvas.drawLine(canvas.getWidth()/2, 0,
canvas.getWidth()/2, canvas.getHeight(), paint);

      final float cornerSize =
Math.min(trackedPos.width(), trackedPos.height())
/ 8.0f;
      float Xmin = (trackedPos.left / 720) * 1280;
      float Ymin = trackedPos.top;
      float Xmax = (trackedPos.right / 720) *
1280;
      float Ymax = trackedPos.bottom;
      trackedPos.set(Xmin, Ymin, Xmax, Ymax);
      canvas.drawRect(trackedPos, boxPaint);

      //variabel referensi jarak
      float jarak2 = 2;
      float jarak3 = 3;
      float jarak4 = 4;
      float jarak5 = 5;
      float ref2 = 603; //(800 / 1080) * 720;
      float ref3 = 450; //(700 / 1080) * 720;
      float ref4 = 337; //(610 / 1080) * 720;
      float ref5 = 316; //(410 / 1080) * 720;

      final Paint paint2 = new Paint();
      paint2.setColor(Color.YELLOW);
      paint2.setStyle(Style.STROKE);
      paint2.setStrokeWidth(0.5f);

      canvas.drawLine(0, ref5,
canvas.getWidth(),ref5, paint2);
      canvas.drawLine(0, ref4,
canvas.getWidth(),ref4, paint2);
      canvas.drawLine(0, ref3,
canvas.getWidth(),ref3, paint2);
      canvas.drawLine(0, ref2,
canvas.getWidth(),ref2, paint2);
```

```
        //perhitungan selisih terkecil
        float S2, S3, S4, S5, Sok = 0, Jok = 0;
        S2 = Math.abs(Ymax - ref2);
        S3 = Math.abs(Ymax - ref3);
        S4 = Math.abs(Ymax - ref4);
        S5 = Math.abs(Ymax - ref5);

        if ((S2 < S3) && (S2 < S4) && (S2 < S5)){
          Sok = ref2;
          Jok = jarak2;
        }
        else if ((S3 < S5) && (S3 < S4) && (S3 <
S2)){
          Sok = ref3;
          Jok = jarak3;
        }
        else if ((S4 < S3) && (S4 < S2) && (S4 <
S5)){
          Sok = ref4;
          Jok = jarak4;
        }
        else if ((S5 < S3) && (S5 < S4) && (S5 <
S2)){
          Sok = ref5;
          Jok = jarak5;
        }

        //perhitungan jarak estimasi
        float jarak_estimasi;
        jarak_estimasi = (Sok * Jok) / Ymax;

//
        Hasil = String.format("%.1f meter",
jarak_estimasi);
        borderedText.drawText(canvas,
trackedPos.left + cornerSize, trackedPos.bottom,
Hasil);

      }
    }

  public static int indexOfSmallest(int[] array){

    // add this
    if (array.length == 0)
      return -1;
```

```
    int index = 0;
    int min = array[index];

    for (int i = 1; i < array.length; i++){
      if (array[i] <= min){
        min = array[i];
        index = i;
      }
    }
    return index;
  }

  private boolean initialized = false;

  public synchronized void onFrame(
          final int w,
          final int h,
          final int rowStride,
          final int sensorOrientation,
          final byte[] frame,
          final long timestamp) {
    if (objectTracker == null && !initialized) {
      ObjectTracker.clearInstance();

      logger.i("Initializing ObjectTracker:
%dx%d", w, h);
      objectTracker = ObjectTracker.getInstance(w,
h, rowStride, true);
      frameWidth = w;
      frameHeight = h;
      this.sensorOrientation = sensorOrientation;
      initialized = true;

      if (objectTracker == null) {
        String message =
                "Object tracking support not
found. "
                        + "See
tensorflow/examples/android/README.md for
details.";
        Toast.makeText(context, message,
Toast.LENGTH_LONG).show();
        logger.e(message);
      }
    }

    if (objectTracker == null) {
```

```
      return;
    }

    objectTracker.nextFrame(frame, null,
timestamp, null, true);

    // Clean up any objects not worth tracking any
more.
    final LinkedList<TrackedRecognition> copyList
=
            new
LinkedList<TrackedRecognition>(trackedObjects);
    for (final TrackedRecognition recognition :
copyList) {
      final ObjectTracker.TrackedObject
trackedObject = recognition.trackedObject;
      final float correlation =
trackedObject.getCurrentCorrelation();
      if (correlation < MIN_CORRELATION) {
        logger.v("Removing tracked object %s
because NCC is %.2f", trackedObject, correlation);
        trackedObject.stopTracking();
        trackedObjects.remove(recognition);

        availableColors.add(recognition.color);
      }
    }
  }

  private void processResults(
          final long timestamp, final
List<Recognition> results, final byte[]
originalFrame) {
    final List<Pair<Float, Recognition>>
rectsToTrack = new LinkedList<Pair<Float,
Recognition>>();

    screenRects.clear();
    final Matrix rgbFrameToScreen = new
Matrix(getFrameToCanvasMatrix());

    for (final Recognition result : results) {
      if (result.getLocation() == null) {
        continue;
      }
      final RectF detectionFrameRect = new
RectF(result.getLocation());
```

```
      final RectF detectionScreenRect = new
RectF();

rgbFrameToScreen.mapRect(detectionScreenRect,
detectionFrameRect);

      logger.v(
              "Result! Frame: " +
result.getLocation() + " mapped to screen:" +
detectionScreenRect);

      screenRects.add(new Pair<Float,
RectF>(result.getConfidence(),
detectionScreenRect));

      if (detectionFrameRect.width() < MIN_SIZE ||
detectionFrameRect.height() < MIN_SIZE) {
        logger.w("Degenerate rectangle! " +
detectionFrameRect);
        continue;
      }

      rectsToTrack.add(new Pair<Float,
Recognition>(result.getConfidence(), result));
    }

    if (rectsToTrack.isEmpty()) {
      logger.v("Nothing to track, aborting.");
      return;
    }

    if (objectTracker == null) {
      trackedObjects.clear();
      for (final Pair<Float, Recognition>
potential : rectsToTrack) {
        final TrackedRecognition
trackedRecognition = new TrackedRecognition();
        trackedRecognition.detectionConfidence =
potential.first;
        trackedRecognition.location = new
RectF(potential.second.getLocation());
        trackedRecognition.trackedObject = null;
        trackedRecognition.title =
potential.second.getTitle();
        trackedRecognition.color =
COLORS[trackedObjects.size()];
```

```
        trackedObjects.add(trackedRecognition);

        if (trackedObjects.size() >=
COLORS.length) {
          break;
        }
      }
      return;
    }

    logger.i("%d rects to track",
rectsToTrack.size());
    for (final Pair<Float, Recognition> potential
: rectsToTrack) {
      handleDetection(originalFrame, timestamp,
potential);
    }
  }

  private void handleDetection(
          final byte[] frameCopy, final long
timestamp, final Pair<Float, Recognition>
potential) {
    final ObjectTracker.TrackedObject
potentialObject =

objectTracker.trackObject(potential.second.getLoca
tion(), timestamp, frameCopy);

    final float potentialCorrelation =
potentialObject.getCurrentCorrelation();
    logger.v(
            "Tracked object went from %s to %s
with correlation %.2f",
            potential.second,
potentialObject.getTrackedPositionInPreviewFrame()
, potentialCorrelation);

    if (potentialCorrelation <
MARGINAL_CORRELATION) {
      logger.v("Correlation too low to begin
tracking %s.", potentialObject);
      potentialObject.stopTracking();
      return;
    }

    final List<TrackedRecognition> removeList =
```

```
new LinkedList<TrackedRecognition>();

    float maxIntersect = 0.0f;

    // This is the current tracked object whose
color we will take. If left null we'll take the
    // first one from the color queue.
    TrackedRecognition recogToReplace = null;

    // Look for intersections that will be
overridden by this object or an intersection that
would
    // prevent this one from being placed.
    for (final TrackedRecognition
trackedRecognition : trackedObjects) {
      final RectF a =
trackedRecognition.trackedObject.getTrackedPositio
nInPreviewFrame();
      final RectF b =
potentialObject.getTrackedPositionInPreviewFrame()
;
      final RectF intersection = new RectF();
      final boolean intersects =
intersection.setIntersect(a, b);

      final float intersectArea =
intersection.width() * intersection.height();
      final float totalArea = a.width() *
a.height() + b.width() * b.height() -
intersectArea;
      final float intersectOverUnion =
intersectArea / totalArea;

      // If there is an intersection with this
currently tracked box above the maximum overlap
      // percentage allowed, either the new
recognition needs to be dismissed or the old
      // recognition needs to be removed and
possibly replaced with the new one.
      if (intersects && intersectOverUnion >
MAX_OVERLAP) {
        if (potential.first <
trackedRecognition.detectionConfidence
                &&
trackedRecognition.trackedObject.getCurrentCorrela
tion() > MARGINAL_CORRELATION) {
          // If track for the existing object is
```

```
still going strong and the detection score was
          // good, reject this new object.
          potentialObject.stopTracking();
          return;
        } else {
          removeList.add(trackedRecognition);

          // Let the previously tracked object
with max intersection amount donate its color to
          // the new object.
          if (intersectOverUnion > maxIntersect) {
            maxIntersect = intersectOverUnion;
            recogToReplace = trackedRecognition;
          }
        }
      }
    }

    // If we're already tracking the max object
and no intersections were found to bump off,
    // pick the worst current tracked object to
remove, if it's also worse than this candidate
    // object.
    if (availableColors.isEmpty() &&
removeList.isEmpty()) {
      for (final TrackedRecognition candidate :
trackedObjects) {
        if (candidate.detectionConfidence <
potential.first) {
          if (recogToReplace == null
                  || candidate.detectionConfidence
< recogToReplace.detectionConfidence) {
            // Save it so that we use this color
for the new object.
            recogToReplace = candidate;
          }
        }
      }
      if (recogToReplace != null) {
        logger.v("Found non-intersecting object to
remove.");
        removeList.add(recogToReplace);
      } else {
        logger.v("No non-intersecting object found
to remove");
      }
    }
```

```
    // Remove everything that got intersected.
    for (final TrackedRecognition
trackedRecognition : removeList) {
      logger.v(
              "Removing tracked object %s with
detection confidence %.2f, correlation %.2f",
              trackedRecognition.trackedObject,

trackedRecognition.detectionConfidence,

trackedRecognition.trackedObject.getCurrentCorrela
tion());

trackedRecognition.trackedObject.stopTracking();
      trackedObjects.remove(trackedRecognition);
      if (trackedRecognition != recogToReplace) {

availableColors.add(trackedRecognition.color);
      }
    }

    if (recogToReplace == null &&
availableColors.isEmpty()) {
      logger.e("No room to track this object,
aborting.");
      potentialObject.stopTracking();
      return;
    }

    // Finally safe to say we can track this
object.
    logger.v(
            "Tracking object %s (%s) with
detection confidence %.2f at position %s",
            potentialObject,
            potential.second.getTitle(),
            potential.first,
            potential.second.getLocation());
    final TrackedRecognition trackedRecognition =
new TrackedRecognition();
    trackedRecognition.detectionConfidence =
potential.first;
    trackedRecognition.trackedObject =
potentialObject;
    trackedRecognition.title =
potential.second.getTitle();
```

```
    // Use the color from a replaced object before
taking one from the color queue.
    trackedRecognition.color =
            recogToReplace != null ?
recogToReplace.color : availableColors.poll();
    trackedObjects.add(trackedRecognition);
  }
}
```

7. **TextToSpeech**

```
public abstract class TextToSpeechActivity extends
CameraActivity implements
TextToSpeech.OnInitListener {
    private TextToSpeech textToSpeech;
    private String lastRecognizedClass="";

    @Override
    public void onInit(int status){
        int result =
textToSpeech.setLanguage(Locale.ENGLISH); //US
        if (result ==
TextToSpeech.LANG_MISSING_DATA
                || result ==
TextToSpeech.LANG_NOT_SUPPORTED) {
            //Log.e(Tag,"Text to speech error:
This Language is not supported");
        }
     else {
        Log.e("message","Text to speech:
Initilization Failed!");
    }
}

    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        textToSpeech = new TextToSpeech(this,
this);
    }

    protected void
speak(List<Classifier.Recognition> results, String
```

```
hasil) {
        if (!(results.isEmpty() ||
lastRecognizedClass.equals(results.get(0).getTitle
())))  {
            lastRecognizedClass = "tangga"+ hasil
;
            if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.LOLLIPOP) {

textToSpeech.speak(lastRecognizedClass,
TextToSpeech.QUEUE_FLUSH, null, null);
            } else {

textToSpeech.speak(lastRecognizedClass,
TextToSpeech.QUEUE_FLUSH, null);
            }
        }
    }

}
```

LEMBAR PERBAIKAN SKRIPSI

"SISTEM DETEKSI DAN ESTIMASI JARAK TANGGA SECARA REALTIME BERBASIS ANDROID UNTUK PENYANDANG TUNANETRA"

OLEH:

HERLINA ANWAR
D121171009

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana tanggal 21 Juli 2022.

Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Dr. Indrabayu, S.T., M.T.,M.Bus.Sys.	
Sekretaris	Dr. Eng. Intan Sari Areni, S.T., M.T.	
Anggota	Prof. Dr. Ir. Andani, M.T.	
	Ir. Christoforus Yohannes, M.T.	

Persetujuan Perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Dr. Indrabayu, S.T., M.T., M.Bus.Sys.	
II	Dr. Eng. Intan Sari Areni, S.T., M.T.	