L A M P I R A N

detect.py

```
source = str(source)
    save_img = not nosave and
not source.endswith(".txt")  #
save inference images
    is_file =
Path(source).suffix[1:] in
(IMG_FORMATS + VID_FORMATS)
    is_url =
source.lower().startswith(("rt
sp://", "rtmp://", "http://",
"https://"))
    webcam =
source.isnumeric() or
source.endswith(".streams") or
(is_url and not is_file)
    screenshot =
source.lower().startswith("scr
een")
    if is_url and is_file:
        source =
check_file(source)  # download

    # Directories
    save_dir =
increment_path(Path(project) /
name, exist_ok=exist_ok)  #
increment run
    (save_dir / "labels" if
save_txt else
save_dir).mkdir(parents=True,
exist_ok=True)  # make dir

    # Load model
    device =
select_device(device)
    model =
DetectMultiBackend(weights,
device=device, dnn=dnn,
data=data, fp16=half)
    stride, names, pt =
model.stride, model.names,
model.pt
    imgsz =
check_img_size(imgsz,
s=stride)  # check image size

    # Dataloader
    bs = 1  # batch_size
    if webcam:
        view_img =
check_imshow(warn=True)
        dataset =
LoadStreams(source,
img_size=imgsz, stride=stride,
auto=pt,
vid_stride=vid_stride)
        bs = len(dataset)
    elif screenshot:
        dataset =
LoadScreenshots(source,
img_size=imgsz, stride=stride,
auto=pt)
    else:
        dataset =
LoadImages(source,
img_size=imgsz, stride=stride,
auto=pt,
vid_stride=vid_stride)
    vid_path, vid_writer =
[None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if
pt or model.triton else bs, 3,
*imgsz))  # warmup
    seen, windows, dt = 0, [],
(Profile(device=device),
Profile(device=device),
Profile(device=device))
    for path, im, im0s,
vid_cap, s in dataset:
        with dt[0]:
```

```
            im =
torch.from_numpy(im).to(model.
device)
            im = im.half() if
model.fp16 else im.float()  #
uint8 to fp16/32
            im /= 255  # 0 -
255 to 0.0 - 1.0
            if len(im.shape)
== 3:
                im = im[None]
# expand for batch dim
            if model.xml and
im.shape[0] > 1:
                ims =
torch.chunk(im, im.shape[0],
0)


        # Inference
        with dt[1]:
            visualize =
increment_path(save_dir /
Path(path).stem, mkdir=True)
if visualize else False
            if model.xml and
im.shape[0] > 1:
                pred = None
                for image in
ims:
                    if pred is
None:
                        pred =
model(image, augment=augment,
visualize=visualize).unsqueeze
(0)
                    else:
                        pred =
torch.cat((pred, model(image,
augment=augment,
visualize=visualize).unsqueeze
(0)), dim=0)
                pred = [pred,
None]
            else:
                pred =
model(im, augment=augment,
visualize=visualize)
        # NMS
        with dt[2]:
            pred =
non_max_suppression(pred,
conf_thres, iou_thres,
classes, agnostic_nms,
max_det=max_det)


        # Second-stage
classifier (optional)
        # pred =
utils.general.apply_classifier
(pred, classifier_model, im,
im0s)


        # Define the path for
the CSV file
        csv_path = save_dir /
"predictions.csv"


        # Create or append to
the CSV file
        def
write_to_csv(image_name,
prediction, confidence):
            """Writes
prediction data for an image
to a CSV file, appending if
the file exists."""
            data = {"Image
Name": image_name,
"Prediction": prediction,
"Confidence": confidence}
            file_exists =
os.path.isfile(csv_path)
            with
open(csv_path, mode="a",
newline="") as f:
                writer =
csv.DictWriter(f,
fieldnames=data.keys())
                if not
file_exists:
```

```
writer.writeheader()

writer.writerow(data)


        # Process predictions
        for i, det in
enumerate(pred):  # per image
            seen += 1
            if webcam:  #
batch_size >= 1
                p, im0, frame
= path[i], im0s[i].copy(),
dataset.count
                s += f"{i}: "
            else:
                p, im0, frame
= path, im0s.copy(),
getattr(dataset, "frame", 0)


            p = Path(p)  # to
Path
            save_path =
str(save_dir / p.name)  #
im.jpg
            txt_path =
str(save_dir / "labels" /
p.stem) + ("" if dataset.mode
== "image" else f"_{frame}")
# im.txt
            s += "{:g}x{:g}
".format(*im.shape[2:])  #
print string
            gn =
torch.tensor(im0.shape)[[1, 0,
1, 0]]  # normalization gain
whwh
            imc = im0.copy()
if save_crop else im0  # for
save_crop
            annotator =
Annotator(im0,
line_width=line_thickness,
example=str(names))
            if len(det):
                # Rescale
boxes from img_size to im0
size
                det[:, :4] =
scale_boxes(im.shape[2:],
det[:, :4], im0.shape).round()


                # Print
results
                for c in
det[:, 5].unique():
                    n =
(det[:, 5] == c).sum()  #
detections per class
                    s += f"{n}
{names[int(c)]}{'s' * (n >
1)}, "  # add to string


                # Write
results
                for *xyxy,
conf, cls in reversed(det):
                    c =
int(cls)  # integer class
                    label =
names[c] if hide_conf else
f"{names[c]}"
                    confidence
= float(conf)

confidence_str =
f"{confidence:.2f}"


                    if
save_csv:

write_to_csv(p.name, label,
confidence_str)


                    if
save_txt:  # Write to file
                        if
save_format == 0:

coords = (
```

```
(xyxy2xywh(torch.tensor(xyxy).
view(1, 4)) / gn).view(-
1).tolist()
                                )
# normalized xywh
                        else:

coords =
(torch.tensor(xyxy).view(1, 4)
/ gn).view(-1).tolist()  #
xyxy
                        line =
(cls, *coords, conf) if
save_conf else (cls, *coords)
# label format
                        with
open(f"{txt_path}.txt", "a")
as f:

f.write(("%g " *
len(line)).rstrip() % line +
"\n")


                    if
save_img or save_crop or
view_img:  # Add bbox to image
                        c =
int(cls)  # integer class
                        label
= None if hide_labels else
(names[c] if hide_conf else
f"{names[c]} {conf:.2f}")

annotator.box_label(xyxy,
label, color=colors(c, True))
                    if
save_crop:

save_one_box(xyxy, imc,
file=save_dir / "crops" /
names[c] / f"{p.stem}.jpg",
BGR=True)


            # Stream results
            im0 =
annotator.result()
            if view_img:
                if
platform.system() == "Linux"
and p not in windows:

windows.append(p)

cv2.namedWindow(str(p),
cv2.WINDOW_NORMAL |
cv2.WINDOW_KEEPRATIO)  # allow
window resize (Linux)

cv2.resizeWindow(str(p),
im0.shape[1], im0.shape[0])

cv2.imshow(str(p), im0)
                cv2.waitKey(1)
# 1 millisecond


            # Save results
(image with detections)
            if save_img:
                if
dataset.mode == "image":

cv2.imwrite(save_path, im0)
                else:  #
'video' or 'stream'
                    if
vid_path[i] != save_path:  #
new video

vid_path[i] = save_path
                        if
isinstance(vid_writer[i],
cv2.VideoWriter):

vid_writer[i].release()  #
release previous video writer
                        if
vid_cap:  # video

fps =
vid_cap.get(cv2.CAP_PROP_FPS)
                            w
=
```

```
int(vid_cap.get(cv2.CAP_PROP_F
RAME_WIDTH))
                                h
=
int(vid_cap.get(cv2.CAP_PROP_F
RAME_HEIGHT))
                            else:
# stream


fps, w, h = 30, im0.shape[1],
im0.shape[0]


save_path =
str(Path(save_path).with_suffi
x(".mp4"))  # force *.mp4
suffix on results videos


vid_writer[i] =
cv2.VideoWriter(save_path,
cv2.VideoWriter_fourcc(*"mp4v"
), fps, (w, h))


vid_writer[i].write(im0)


        # Print time
(inference-only)
        LOGGER.info(f"{s}{''
if len(det) else '(no
detections), '}{dt[1].dt *
1e3:.1f}ms")
    # Print results
    t = tuple(x.t / seen * 1e3
for x in dt)  # speeds per
image
    LOGGER.info(f"Speed:
%.1fms pre-process, %.1fms
inference, %.1fms NMS per
image at shape {(1, 3,
*imgsz)}" % t)
    if save_txt or save_img:
        s =
f"\n{len(list(save_dir.glob('l
abels/*.txt')))} labels saved
to {save_dir / 'labels'}" if
save_txt else ""
        LOGGER.info(f"Results
saved to {colorstr('bold',
save_dir)}{s}")
    if update:


strip_optimizer(weights[0])  #
update model (to fix
SourceChangeWarning)



def parse_opt()
```

detect2.py

```
import argparse
import csv
import os
import platform
import sys
from pathlib import Path
import pathlib
temp = pathlib.PosixPath
pathlib.PosixPath =
pathlib.WindowsPath
import torch


FILE =
Path(__file__).resolve()
ROOT = FILE.parents[0]  #
YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))
# add ROOT to PATH
ROOT =
Path(os.path.relpath(ROOT,
Path.cwd()))  # relative


from
ultralytics.utils.plotting
import Annotator, colors,
save_one_box


from models.common import
DetectMultiBackend
from utils.dataloaders import
IMG_FORMATS, VID_FORMATS,
LoadImages, LoadScreenshots,
LoadStreams
from utils.general import (
    LOGGER,
    Profile,
    check_file,
    check_img_size,
    check_imshow,
    check_requirements,
    colorstr,
    cv2,
    increment_path,
    non_max_suppression,
    print_args,
    scale_boxes,
    strip_optimizer,
    xyxy2xywh,
)
from utils.torch_utils import
select_device,
smart_inference_mode



@smart_inference_mode()
def run(
    weights=ROOT / "best.pt",
# model path or triton URL
    source=ROOT /
"data/images",  #
file/dir/URL/glob/screen/0(web
cam)
    data=ROOT /
"data/coco128.yaml",  #
dataset.yaml path
    imgsz=(640, 640),  #
inference size (height, width)
    conf_thres=0.25,  #
confidence threshold
    iou_thres=0.45,  # NMS IOU
threshold
    max_det=1000,  # maximum
detections per image
    device="",  # cuda device,
i.e. 0 or 0,1,2,3 or cpu
    view_img=False,  # show
results
```

```
    save_txt=False,  # save
results to *.txt

    save_format=0,  # save
boxes coordinates in YOLO
format or Pascal-VOC format (0
for YOLO and 1 for Pascal-VOC)

    save_csv=False,  # save
results in CSV format

    save_conf=False,  # save
confidences in --save-txt
labels

    save_crop=False,  # save
cropped prediction boxes

    nosave=False,  # do not
save images/videos

    classes=None,  # filter by
class: --class 0, or --class 0
2

    agnostic_nms=False,  #
class-agnostic NMS

    augment=False,  #
augmented inference

    visualize=False,  #
visualize features

    update=False,  # update
all models

    project=ROOT /
"runs/detect",  # save results
to project/name

    name="exp",  # save
results to project/name

    exist_ok=False,  #
existing project/name ok, do
not increment

    line_thickness=3,  #
bounding box thickness
(pixels)

    hide_labels=False,  # hide
labels

    hide_conf=False,  # hide
confidences

    half=False,  # use FP16
half-precision inference

    dnn=False,  # use OpenCV
DNN for ONNX inference

    vid_stride=1,  # video
frame-rate stride

):

    """

    Runs YOLOv5 detection
inference on various sources
like images, videos,
directories, streams, etc.


    Args:

        weights (str | Path):
Path to the model weights file
or a Triton URL. Default is
'yolov5s.pt'.

        source (str | Path):
Input source, which can be a
file, directory, URL, glob
pattern, screen capture, or
webcam

            index. Default is
'data/images'.

        data (str | Path):
Path to the dataset YAML file.
Default is
'data/coco128.yaml'.

        imgsz (tuple[int,
int]): Inference image size as
a tuple (height, width).
Default is (640, 640).

        conf_thres (float):
Confidence threshold for
detections. Default is 0.25.

        iou_thres (float):
Intersection Over Union (IOU)
threshold for non-max
suppression. Default is 0.45.

        max_det (int): Maximum
number of detections per
image. Default is 1000.

        device (str): CUDA
device identifier (e.g., '0'
or '0,1,2,3') or 'cpu'.
Default is an empty string,
which uses the

            best available
device.

        view_img (bool): If
True, display inference
```

```
results using OpenCV. Default
is False.

        save_txt (bool): If
True, save results in a text
file. Default is False.

        save_csv (bool): If
True, save results in a CSV
file. Default is False.

        save_conf (bool): If
True, include confidence
scores in the saved results.
Default is False.

        save_crop (bool): If
True, save cropped prediction
boxes. Default is False.

        nosave (bool): If
True, do not save inference
images or videos. Default is
False.

        classes (list[int]):
List of class indices to
filter detections by. Default
is None.

        agnostic_nms (bool):
If True, perform class-
agnostic non-max suppression.
Default is False.

        augment (bool): If
True, use augmented inference.
Default is False.

        visualize (bool): If
True, visualize feature maps.
Default is False.

        update (bool): If
True, update all models'
weights. Default is False.

        project (str | Path):
Directory to save results.
Default is 'runs/detect'.

        name (str): Name of
the current experiment; used
to create a subdirectory
within 'project'. Default is
'exp'.

        exist_ok (bool): If
True, existing directories
with the same name are reused
instead of being incremented.
Default is
            False.

        line_thickness (int):
Thickness of bounding box
lines in pixels. Default is 3.

        hide_labels (bool): If
True, do not display labels on
bounding boxes. Default is
False.

        hide_conf (bool): If
True, do not display
confidence scores on bounding
boxes. Default is False.

        half (bool): If True,
use FP16 half-precision
inference. Default is False.

        dnn (bool): If True,
use OpenCV DNN backend for
ONNX inference. Default is
False.

        vid_stride (int):
Stride for processing video
frames, to skip frames between
processing. Default is 1.


    Returns:

        None


    Examples:

        ```python

        from ultralytics
import run


        # Run inference on an
image

run(source='data/images/exampl
e.jpg', weights='yolov5s.pt',
device='0')


        # Run inference on a
video with specific confidence
threshold

run(source='data/videos/exampl
e.mp4', weights='yolov5s.pt',
conf_thres=0.4, device='0')
```
```

```
        ```
    """
    source = str(source)
    save_img = not nosave and
not source.endswith(".txt") #
save inference images
    is_file =
Path(source).suffix[1:] in
(IMG_FORMATS + VID_FORMATS)
    is_url =
source.lower().startswith(("rt
sp://", "rtmp://", "http://",
"https://"))
    webcam =
source.isnumeric() or
source.endswith(".streams") or
(is_url and not is_file)
    screenshot =
source.lower().startswith("scr
een")
    if is_url and is_file:
        source =
check_file(source) # download

    # Directories
    save_dir =
increment_path(Path(project) /
name, exist_ok=exist_ok) #
increment run
    (save_dir / "labels" if
save_txt else
save_dir).mkdir(parents=True,
exist_ok=True) # make dir

    # Load model
    device =
select_device(device)
    model =
DetectMultiBackend(weights,
device=device, dnn=dnn,
data=data, fp16=half)
    stride, names, pt =
model.stride, model.names,
model.pt
    imgsz =
check_img_size(imgsz,
s=stride) # check image size

    # Dataloader
    bs = 1 # batch_size
    if webcam:
        view_img =
check_imshow(warn=True)
        dataset =
LoadStreams(source,
img_size=imgsz, stride=stride,
auto=pt,
vid_stride=vid_stride)
        bs = len(dataset)
    elif screenshot:
        dataset =
LoadScreenshots(source,
img_size=imgsz, stride=stride,
auto=pt)
    else:
        dataset =
LoadImages(source,
img_size=imgsz, stride=stride,
auto=pt,
vid_stride=vid_stride)
    vid_path, vid_writer =
[None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if
pt or model.triton else bs, 3,
*imgsz)) # warmup
    seen, windows, dt = 0, [],
(Profile(device=device),
Profile(device=device),
Profile(device=device))
    for path, im, im0s,
vid_cap, s in dataset:
        with dt[0]:
            im =
torch.from_numpy(im).to(model.
device)
            im = im.half() if
model.fp16 else im.float() #
uint8 to fp16/32
```
```

```
            im /= 255  # 0 - 255 to 0.0 - 1.0
            if len(im.shape) == 3:
                im = im[None] # expand for batch dim
            if model.xml and im.shape[0] > 1:
                ims = torch.chunk(im, im.shape[0], 0)


        # Inference
        with dt[1]:
            visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else False
            if model.xml and im.shape[0] > 1:
                pred = None
                for image in ims:
                    if pred is None:
                        pred = model(image, augment=augment, visualize=visualize).unsqueeze(0)
                    else:
                        pred = torch.cat((pred, model(image, augment=augment, visualize=visualize).unsqueeze(0)), dim=0)
                pred = [pred, None]
            else:
                pred = model(im, augment=augment, visualize=visualize)
        # NMS
        with dt[2]:
            pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)


        # Second-stage classifier (optional)
        # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)


        # Define the path for the CSV file
        csv_path = save_dir / "predictions.csv"


        # Create or append to the CSV file
        def write_to_csv(image_name, prediction, confidence):
            """Writes prediction data for an image to a CSV file, appending if the file exists."""
            data = {"Image Name": image_name, "Prediction": prediction, "Confidence": confidence}
            file_exists = os.path.isfile(csv_path)
            with open(csv_path, mode="a", newline="") as f:
                writer = csv.DictWriter(f, fieldnames=data.keys())
                if not file_exists:

writer.writeheader()

writer.writerow(data)


        # Process predictions
```

```
        for i, det in
enumerate(pred):  # per image
            seen += 1
            if webcam:  #
batch_size >= 1
                p, im0, frame
= path[i], im0s[i].copy(),
dataset.count
                s += f"{i}: "
            else:
                p, im0, frame
= path, im0s.copy(),
getattr(dataset, "frame", 0)


            p = Path(p)  # to
Path
            save_path =
str(save_dir / p.name)  #
im.jpg
            txt_path =
str(save_dir / "labels" /
p.stem) + ("" if dataset.mode
== "image" else f"_{frame}")
# im.txt
            s += "{:g}x{:g}
".format(*im.shape[2:])  #
print string
            gn =
torch.tensor(im0.shape)[[1, 0,
1, 0]]  # normalization gain
whwh
            imc = im0.copy()
if save_crop else im0  # for
save_crop
            annotator =
Annotator(im0,
line_width=line_thickness,
example=str(names))
            if len(det):
                # Rescale
boxes from img_size to im0
size
                det[:, :4] =
scale_boxes(im.shape[2:],
det[:, :4], im0.shape).round()
                # Print
results
                for c in
det[:, 5].unique():
                    n =
(det[:, 5] == c).sum()  #
detections per class
                    s += f"{n}
{names[int(c)]}{'s' * (n >
1)}, "  # add to string


                # Write
results
                for *xyxy,
conf, cls in reversed(det):
                    c =
int(cls)  # integer class
                    label =
names[c] if hide_conf else
f"{names[c]}"
                    confidence
= float(conf)

confidence_str =
f"{confidence:.2f}"


                    if
save_csv:

write_to_csv(p.name, label,
confidence_str)


                    if
save_txt:  # Write to file
                        if
save_format == 0:

coords = (

(xyxy2xywh(torch.tensor(xyxy).
view(1, 4)) / gn).view(-
1).tolist()
                            )
# normalized xywh
                        else:
```

```
coords =
(torch.tensor(xyxy).view(1, 4)
/ gn).view(-1).tolist()  #
xyxy

                        line =
(cls, *coords, conf) if
save_conf else (cls, *coords)
# label format

                        with
open(f"{txt_path}.txt", "a")
as f:


f.write(("%g " *
len(line)).rstrip() % line +
"\n")


                    if
save_img or save_crop or
view_img:  # Add bbox to image

                        c =
int(cls)  # integer class

                        label
= None if hide_labels else
(names[c] if hide_conf else
f"{names[c]} {conf:.2f}")


annotator.box_label(xyxy,
label, color=colors(c, True))

                    if
save_crop:


save_one_box(xyxy, imc,
file=save_dir / "crops" /
names[c] / f"{p.stem}.jpg",
BGR=True)


            # Stream results

            im0 =
annotator.result()

            if view_img:

                if
platform.system() == "Linux"
and p not in windows:


windows.append(p)

cv2.namedWindow(str(p),
cv2.WINDOW_NORMAL |
cv2.WINDOW_KEEPRATIO)  # allow
window resize (Linux)


cv2.resizeWindow(str(p),
im0.shape[1], im0.shape[0])


cv2.imshow(str(p), im0)

                cv2.waitKey(1)
# 1 millisecond


            # Save results
(image with detections)

            if save_img:

                if
dataset.mode == "image":


cv2.imwrite(save_path, im0)

                else:  #
'video' or 'stream'

                    if
vid_path[i] != save_path:  #
new video


vid_path[i] = save_path

                        if
isinstance(vid_writer[i],
cv2.VideoWriter):


vid_writer[i].release()  #
release previous video writer

                        if
vid_cap:  # video


fps =
vid_cap.get(cv2.CAP_PROP_FPS)

                            w
=
int(vid_cap.get(cv2.CAP_PROP_F
RAME_WIDTH))

                            h
=
int(vid_cap.get(cv2.CAP_PROP_F
RAME_HEIGHT))
```

```
                        else:  # stream
fps, w, h = 30, im0.shape[1], im0.shape[0]
save_path = str(Path(save_path).with_suffix(".mp4"))  # force *.mp4 suffix on results videos
vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*"mp4v"), fps, (w, h))
vid_writer[i].write(im0)

        # Print time (inference-only)
        LOGGER.info(f"{s}{'' if len(det) else '(no detections), '}{dt[1].dt * 1e3:.1f}ms")

    # Print results
    t = tuple(x.t / seen * 1e3 for x in dt)  # speeds per image
    LOGGER.info(f"Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image at shape {(1, 3, *imgsz)}" % t)
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ""
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    if update:
strip_optimizer(weights[0])  # update model (to fix SourceChangeWarning)


def parse_opt():
```

ekstraksi.py

```
import tkinter as tk
from tkinter import filedialog, ttk
import cv2
from PIL import Image, ImageTk
import subprocess
import time
import os
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np


# ==========================
# Fungsi untuk tombol animasi
# ==========================
class AnimatedButton(tk.Canvas):
    def __init__(self, master, text, command=None, bg="#4CAF50", fg="white", activebg="#45a049", width=200, height=50, icon=None):

super().__init__(master, width=width, height=height, highlightthickness=0, bg=master["bg"])
        self.command = command
        self.bg = bg
        self.fg = fg
        self.activebg = activebg
        self.width = width
        self.height = height
        self.text = text
        self.icon = icon
        self.current_color = bg

        # Buat tombol kotak rounded
        self.rect = self.create_rounded_rect(5, 5, width-5, height-5, 20, fill=bg, outline="")
        self.label = self.create_text(width//2, height//2, text=(icon + " " if icon else "") + text,

fill=fg, font=("Helvetica", 11, "bold"))

        # Event binding
        self.bind("<Enter>", self.on_hover)
        self.bind("<Leave>", self.on_leave)
        self.bind("<Button-1>", self.on_click)

    def create_rounded_rect(self, x1, y1, x2, y2, r=25, **kwargs):
        points = [
            x1+r, y1,
            x1+r, y1,
            x2-r, y1,
            x2-r, y1,
            x2, y1,
            x2, y1+r,
            x2, y1+r,
            x2, y2-r,
            x2, y2-r,
            x2, y2,
            x2-r, y2,
```

```
            x2-r, y2,
            x1+r, y2,
            x1+r, y2,
            x1, y2,
            x1, y2-r,
            x1, y2-r,
            x1, y1+r,
            x1, y1+r,
            x1, y1,
        ]
        return
self.create_polygon(points,
smooth=True, **kwargs)


    def animate_color(self,
target_color):
        # Animasi transisi
warna
        start_rgb =
self.winfo_rgb(self.current_co
lor)
        end_rgb =
self.winfo_rgb(target_color)
        steps = 10
        for i in range(1,
steps+1):
            r =
int(start_rgb[0] +
(end_rgb[0]-start_rgb[0]) *
i/steps) // 256
            g =
int(start_rgb[1] +
(end_rgb[1]-start_rgb[1]) *
i/steps) // 256
            b =
int(start_rgb[2] +
(end_rgb[2]-start_rgb[2]) *
i/steps) // 256
            color =
f"#{r:02x}{g:02x}{b:02x}"
            self.after(i*10,
lambda c=color:
self.itemconfig(self.rect,
fill=c))
        self.current_color =
target_color


    def on_hover(self, event):

self.animate_color(self.active
bg)


    def on_leave(self, event):

self.animate_color(self.bg)


    def on_click(self, event):
        if self.command:
            self.command()


# ==========================
# Fungsi-fungsi utama (OPEN,
YOLO, dsb)
# ==========================


def open_image():
    global filepath,
img_resized
    filepath =
filedialog.askopenfilename(fil
etypes=[("Image files",
"*.jpg;*.png;*.jpeg")])
    if not filepath:
        return
    img = cv2.imread(filepath)
    img_rgb =
cv2.cvtColor(img,
cv2.COLOR_BGR2RGB)
    img_resized =
cv2.resize(img_rgb, (300,
300))  # Resize image agar pas
    img_pil =
Image.fromarray(img_resized)
    img_tk =
ImageTk.PhotoImage(img_pil)
    panel.config(image=img_tk)
```

```
    panel.image = img_tk

result_label.config(text="Gamb
ar berhasil dimuat. Klik
gambar untuk melihat nilai
piksel.")
    clear_histogram()
    clear_table()


def
get_latest_exp_folder(base_pat
h):
    subfolders =
[os.path.join(base_path, f)
for f in os.listdir(base_path)
if
os.path.isdir(os.path.join(bas
e_path, f))]
    if not subfolders:
        return None
    latest_folder =
max(subfolders,
key=os.path.getctime)
    return latest_folder


def identify_corn():
    global img_resized
    if not filepath:

result_label.config(text="Gaga
l! Tidak ada gambar yang
dimuat.")
        return


    detect_script = r"E:/Data
Skripsi ku/Deteksi Jenis
Jagung/yolo/yolov5/content/yol
ov5/detect2.py"
    result_folder_base =
r"E:/Data Skripsi ku/Deteksi
Jenis
Jagung/yolo/yolov5/content/yol
ov5/runs/detect"

result_label.config(text="Seda
ng mendeteksi... Mohon
tunggu.")
    root.update_idletasks()


    subprocess.run(["python",
detect_script, "--source",
filepath])
    time.sleep(3)


    latest_folder =
get_latest_exp_folder(result_f
older_base)
    if not latest_folder:

result_label.config(text="Gaga
l menemukan folder hasil
deteksi.")
        return


    detected_image_path = None
    for file in
os.listdir(latest_folder):
        if
file.endswith((".jpg",
".png")):

detected_image_path =
os.path.join(latest_folder,
file)
            break


    if detected_image_path and
os.path.exists(detected_image_
path):
        img_detected =
cv2.imread(detected_image_path
)
        img_detected =
cv2.cvtColor(img_detected,
cv2.COLOR_BGR2RGB)
```

```
        img_resized =
cv2.resize(img_detected, (400,
400))

        img_pil =
Image.fromarray(img_resized)
        img_tk =
ImageTk.PhotoImage(img_pil)

panel.config(image=img_tk)
        panel.image = img_tk

result_label.config(text="Dete
ksi selesai! Klik gambar untuk
lihat nilai piksel.")
        clear_histogram()
        clear_table()
    else:

result_label.config(text="Gaga
l menampilkan hasil deteksi.")

def zoom_image(event):
    global img_resized
    zoom_factor = 1.1 if
event.delta > 0 else 0.9
    new_width =
int(img_resized.shape[1] *
zoom_factor)
    new_height =
int(img_resized.shape[0] *
zoom_factor)

    img_resized_zoomed =
cv2.resize(img_resized,
(new_width, new_height))
    img_pil =
Image.fromarray(img_resized_zo
omed)
    img_tk =
ImageTk.PhotoImage(img_pil)

    panel.config(image=img_tk)
    panel.image = img_tk

def show_pixel_value(event):
    global img_resized,
canvas_hist, tree
    if img_resized is None:
        return
    x, y = event.x, event.y
    if x <
img_resized.shape[1] and y <
img_resized.shape[0]:
        pixel = img_resized[y,
x]  # format RGB

        clear_table()
        tree =
ttk.Treeview(right_frame,
columns=("Channel", "Nilai"),
show="headings", height=3)

tree.heading("Channel",
text="Channel")
        tree.heading("Nilai",
text="Nilai")
        tree.insert("", "end",
values=("Red", pixel[0]))
        tree.insert("", "end",
values=("Green", pixel[1]))
        tree.insert("", "end",
values=("Blue", pixel[2]))
        tree.grid(row=0,
column=0, pady=5)

        half_size = 5
        y1, y2 = max(0, y-
half_size),
min(img_resized.shape[0],
y+half_size)
        x1, x2 = max(0, x-
half_size),
min(img_resized.shape[1],
x+half_size)
        region =
img_resized[y1:y2, x1:x2]
```

```
        fig, ax =
plt.subplots(figsize=(3.5,2.5)
, dpi=100)
        colors = ('r','g','b')
        labels =
("Red","Green","Blue")
        for i,col in
enumerate(colors):
            hist =
cv2.calcHist([region],[i],None
,[256],[0,256])

ax.bar(np.arange(256),
hist.flatten(), color=col,
alpha=0.5, label=labels[i])
        ax.set_xlim([0,256])

ax.set_title("Histogram Area
10x10 Piksel")

ax.set_xlabel("Intensitas")
        ax.set_ylabel("Jumlah
Piksel")
        ax.legend()


        clear_histogram()
        canvas_hist =
FigureCanvasTkAgg(fig,
master=right_frame)
        canvas_hist.draw()

canvas_hist.get_tk_widget().gr
id(row=1, column=0, pady=5)


def clear_histogram():
    global canvas_hist
    if canvas_hist is not
None:

canvas_hist.get_tk_widget().de
stroy()
        canvas_hist = None


def clear_table():
    global tree
    if tree is not None:
        tree.destroy()
        tree = None


# ==========================
# MAIN WINDOW
# ==========================
root = tk.Tk()
root.title("Identifikasi Jenis
Jagung")
root.geometry("950x550")
root.configure(bg="#f5f5f5")


filepath = ""
img_resized = None
canvas_hist = None
tree = None


main_frame = tk.Frame(root,
bg="#f5f5f5")
main_frame.pack(fill="both",
expand=True)


left_frame =
tk.Frame(main_frame,
bg="#f5f5f5")
left_frame.grid(row=0,
column=0, padx=10, pady=10)


right_frame =
tk.Frame(main_frame,
bg="#f5f5f5")
right_frame.grid(row=0,
column=1, padx=10, pady=10,
sticky="n")


# Tombol pakai animasi
btn_open =
AnimatedButton(root,
```

```
text="Buka Citra",
command=open_image,
bg="#4CAF50",
activebg="#45a049", icon="")
btn_open.pack(pady=5)


panel = tk.Label(left_frame,
bg="#ddd")
panel.pack()


btn_identify =
AnimatedButton(root,
text="Identifikasi Jagung",
command=identify_corn,
bg="#2196F3",
activebg="#0b7dda", icon="")
btn_identify.pack(pady=5)


result_label = tk.Label(root,
text="Pilih gambar untuk
memulai.", font=("Helvetica",
12), wraplength=500,
bg="#f5f5f5")
result_label.pack(pady=5)


panel.bind("<MouseWheel>",
zoom_image)
panel.bind("<Button-1>",
show_pixel_value)


root.mainloop()
```

GUI.py

```
import tkinter as tk
from tkinter import PhotoImage
import subprocess


# Fungsi untuk menjalankan deteksi.py
def run_deteksi():
    subprocess.run(["python", "E:/Data Skripsi ku/Deteksi Jenis Jagung/yolo/yolov5/content/yolov5/ekstraksi.py"])


# Fungsi untuk menjalankan realtimedeteksi.py
def run_realtime():
    subprocess.run(["python", "detect.py"])


# Fungsi untuk logout
def logout():
    root.destroy()


# Membuat jendela utama
root = tk.Tk()
root.title("Deteksi Jenis Jagung")


root.geometry("800x600")


# Menambahkan gambar background
bg_image = PhotoImage(file="E:/Data Skripsi ku/Deteksi Jenis Jagung/yolo/yolov5/content/yolov5/jagung1.png")  # Pastikan file background.png ada
bg_label = tk.Label(root, image=bg_image)
bg_label.place(relwidth=1, relheight=1)  # Menampilkan gambar pada seluruh area


# Menambahkan judul di atas gambar
title = tk.Label(root, text="Deteksi Jenis-Jenis Jagung", font=("Helvetica", 24, "bold"), fg="white", bg="#000000", padx=10, pady=10)
title.pack(pady=20)

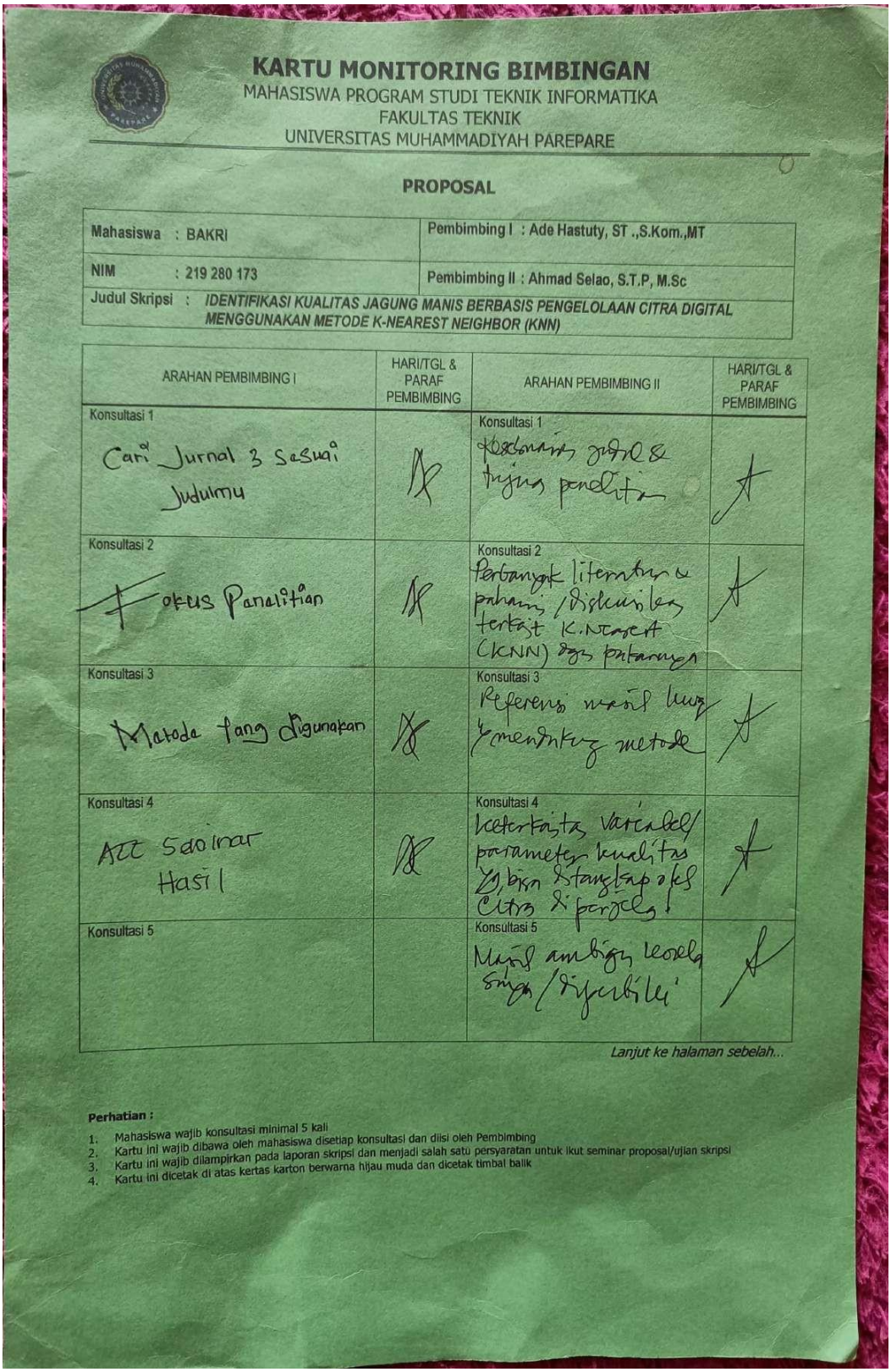

# Menambahkan tombol untuk Real-time Deteksi
button_realtime = tk.Button(root, text="Camera Testing", font=("Helvetica", 14), bg="blue", fg="white", command=run_realtime)
button_realtime.pack(pady=10)


# Menambahkan tombol untuk Deteksi
button_deteksi = tk.Button(root, text="Picture Testing", font=("Helvetica", 14), bg="green", fg="white", command=run_deteksi)
button_deteksi.pack(pady=10)


# Menambahkan tombol untuk Log Out
button_logout = tk.Button(root, text="Log Out", font=("Helvetica", 14), bg="red", fg="white", command=logout)
button_logout.pack(pady=10)


# Menjalankan aplikasi Tkinter
root.mainloop(
```

Lampiran - 2 Kartu Minotoring Bimbingan Proposal



KARTU MONITORING BIMBINGAN
MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

PROPOSAL

Mahasiswa : BAKRI	Pembimbing I : Ade Hastuty, ST .,S.Kom.,MT
NIM : 219 280 173	Pembimbing II : Ahmad Selao, S.T.P, M.Sc
Judul Skripsi : *IDENTIFIKASI KUALITAS JAGUNG MANIS BERBASIS PENGELOLAAN CITRA DIGITAL MENGGUNAKAN METODE K-NEAREST NEIGHBOR (KNN)*	

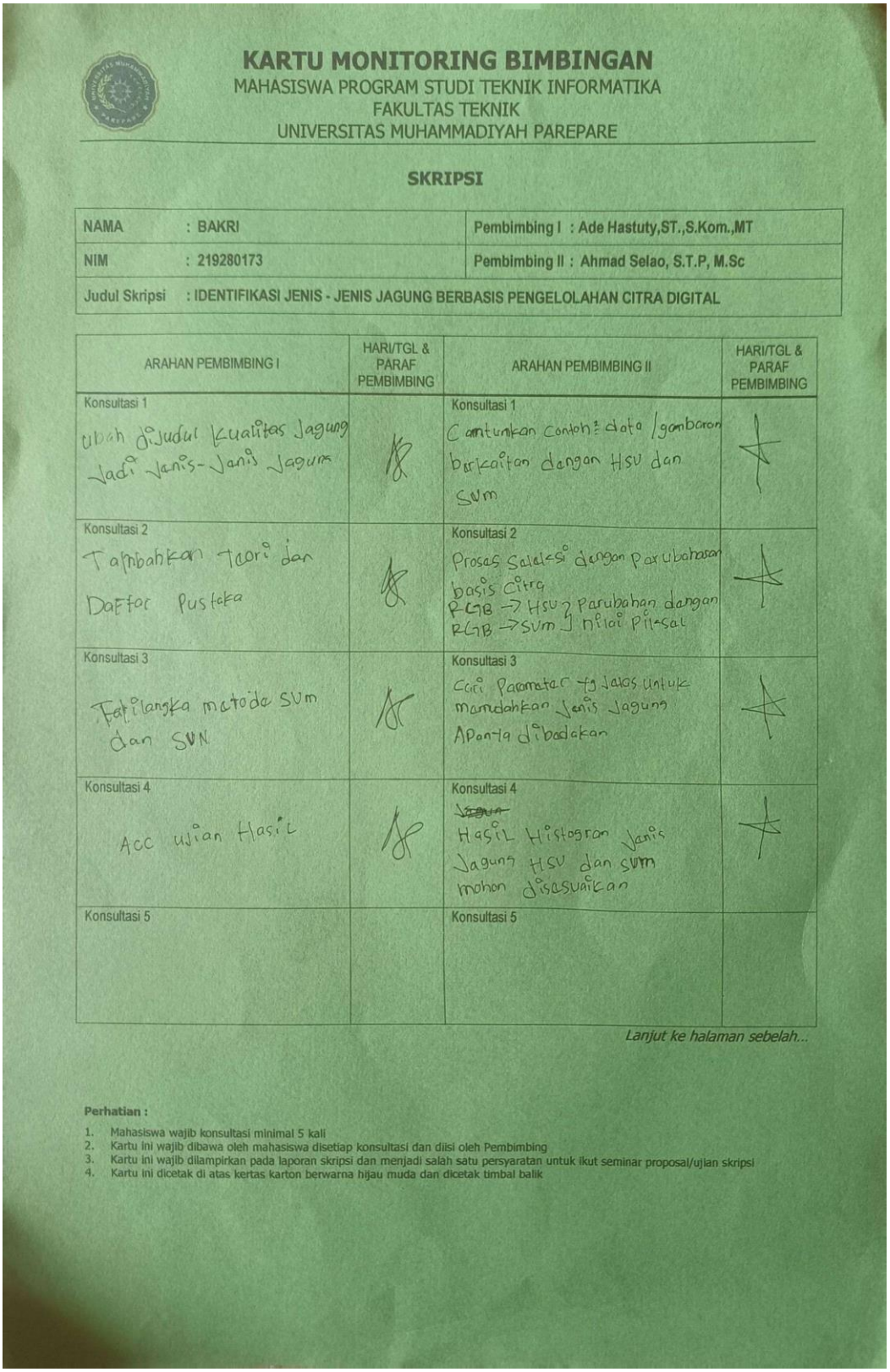
ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Cari Jurnal 3 sesuai Judulmu		Konsultasi 1 Perbanyak judul & tujuan penelitian	
Konsultasi 2 Fokus Penalitian		Konsultasi 2 Perbanyak literatur & pahami / diskusikan terkait K-Nearest (KNN) dgn pakarnya	
Konsultasi 3 Metode yang digunakan		Konsultasi 3 Referensi masih kurang mendukung metode	
Konsultasi 4 ACC Seminar Hasil		Konsultasi 4 ketertarikan variabel/ parameter kualitas yg bisa ditangkap oleh citra di pergelas!	
Konsultasi 5		Konsultasi 5 Masih ambigu, coba lagi/diperbaiki	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

Lampiran - 3 Kartu Minotoring Bimbingan Hasil



KARTU MONITORING BIMBINGAN
MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

SKRIPSI

NAMA	: BAKRI	Pembimbing I	: Ade Hastuty,ST.,S.Kom.,MT
NIM	: 219280173	Pembimbing II	: Ahmad Selao, S.T.P, M.Sc
Judul Skripsi	: IDENTIFIKASI JENIS - JENIS JAGUNG BERBASIS PENGELOLAHAN CITRA DIGITAL		

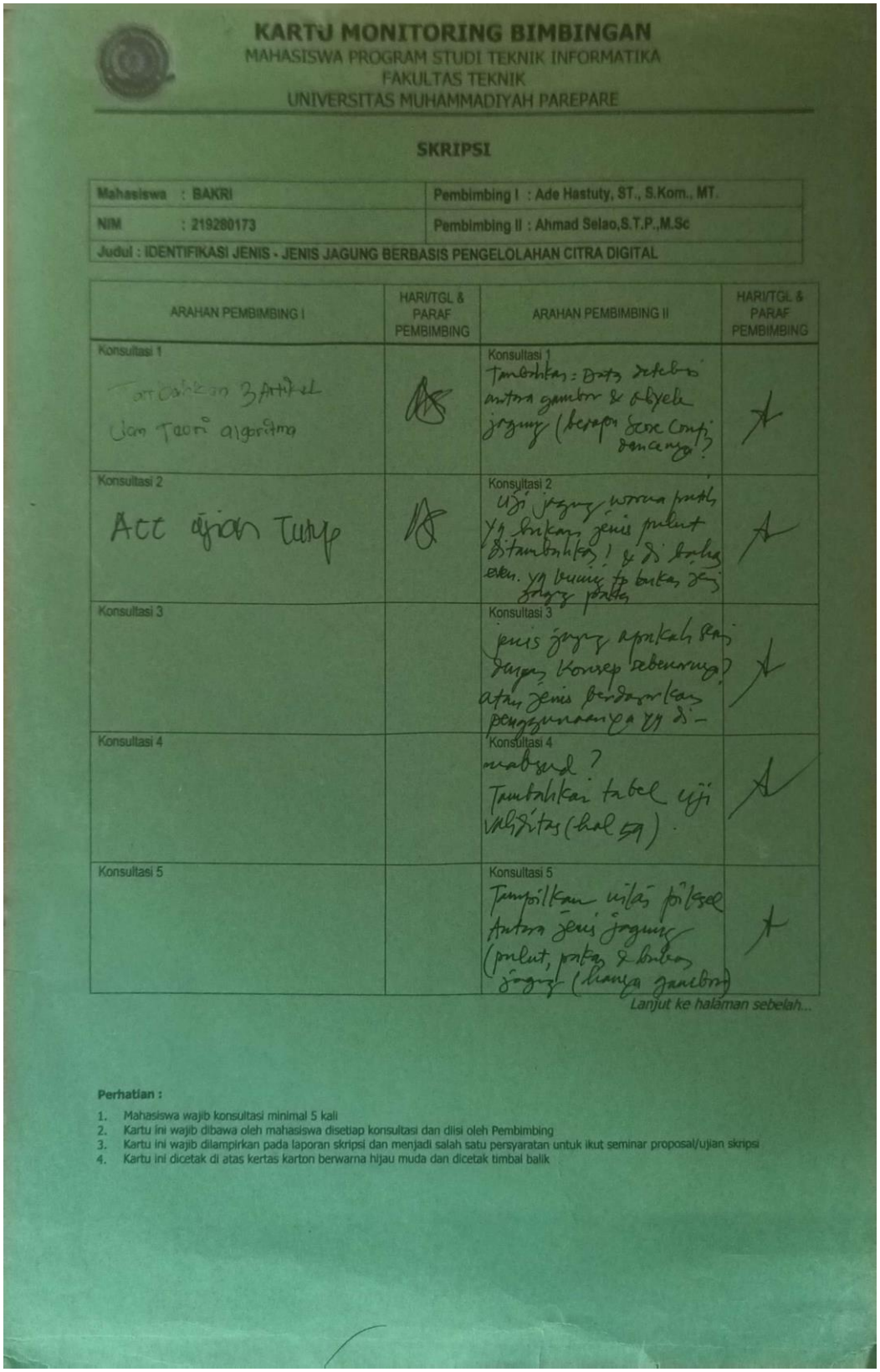
ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Ubah dijudul kualitas jagung jadi Jenis-Jenis Jagung		Konsultasi 1 Cantumkan contoh2 data /gambaran berkaitan dengan HSV dan SVM	
Konsultasi 2 Tambahkan Teori dan Daftar Pustaka		Konsultasi 2 Prosas Seleksi dengan perubahan basis citra RGB → HSV } Perubahan dengan RGB → SVM } nilai pixsal	
Konsultasi 3 Tahaplangka metode SVM dan SVN		Konsultasi 3 Cari Parameter yg jelas untuk mendahkan Jenis Jagung Apanya dibadakan	
Konsultasi 4 Acc ujian Hasil		Konsultasi 4 ~~Jagung~~ Hasil Histogran Jenis Jagung HSV dan SVM mohon disesuaikan	
Konsultasi 5		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

Lampiran - 4 Kartu Minotoring Bimbingan Skripsi



KARTU MONITORING BIMBINGAN
MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

SKRIPSI

Mahasiswa : BAKRI	Pembimbing I : Ade Hastuty, ST., S.Kom., MT.
NIM : 219280173	Pembimbing II : Ahmad Selao,S.T.P.,M.Sc
Judul : IDENTIFIKASI JENIS - JENIS JAGUNG BERBASIS PENGELOLAHAN CITRA DIGITAL	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Tambahkan 3 Artikel dan Teori algoritma		Konsultasi 1 Tambahkan : Data sebelum antara gambar & objek jagung (berapa skor confi dancenya?	
Konsultasi 2 Acc ujian Tutup		Konsultasi 2 Uji jagung warna putih, yg bukan jenis pulut ditambahkan ! & di bahas eks. yg warna to bukan jenis jagung pulut	
Konsultasi 3		Konsultasi 3 jenis jagung apakah saja dapat konsep sebenarnya? atau jenis berdasarkan penggunaanya yg di-	
Konsultasi 4		Konsultasi 4 realisasi ? Tambahkan tabel uji validitas (hal 59)	
Konsultasi 5		Konsultasi 5 Tampilkan nilai pixel antara jenis jagung (pulut, putih & bukan jagung (hanya gambar)	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

Lampiran - 5 Sertifikat Laboratorium Praktikum



SERTIFIKAT

Nomor : 510/PRK/LB-TI/II.3.AU/D/2023

Diberikan Kepada :

Bakri

NIM. 21928017 3

Telah Menyelesaikan

PRAKTIKUM V

Laboratorium Teknik Informatika

Program Studi Teknik Informatika Fakultas Teknik

Universitas Muhammadiyah Parepare

Pada tanggal 16 Januari 2023 - 29 Maret 2023

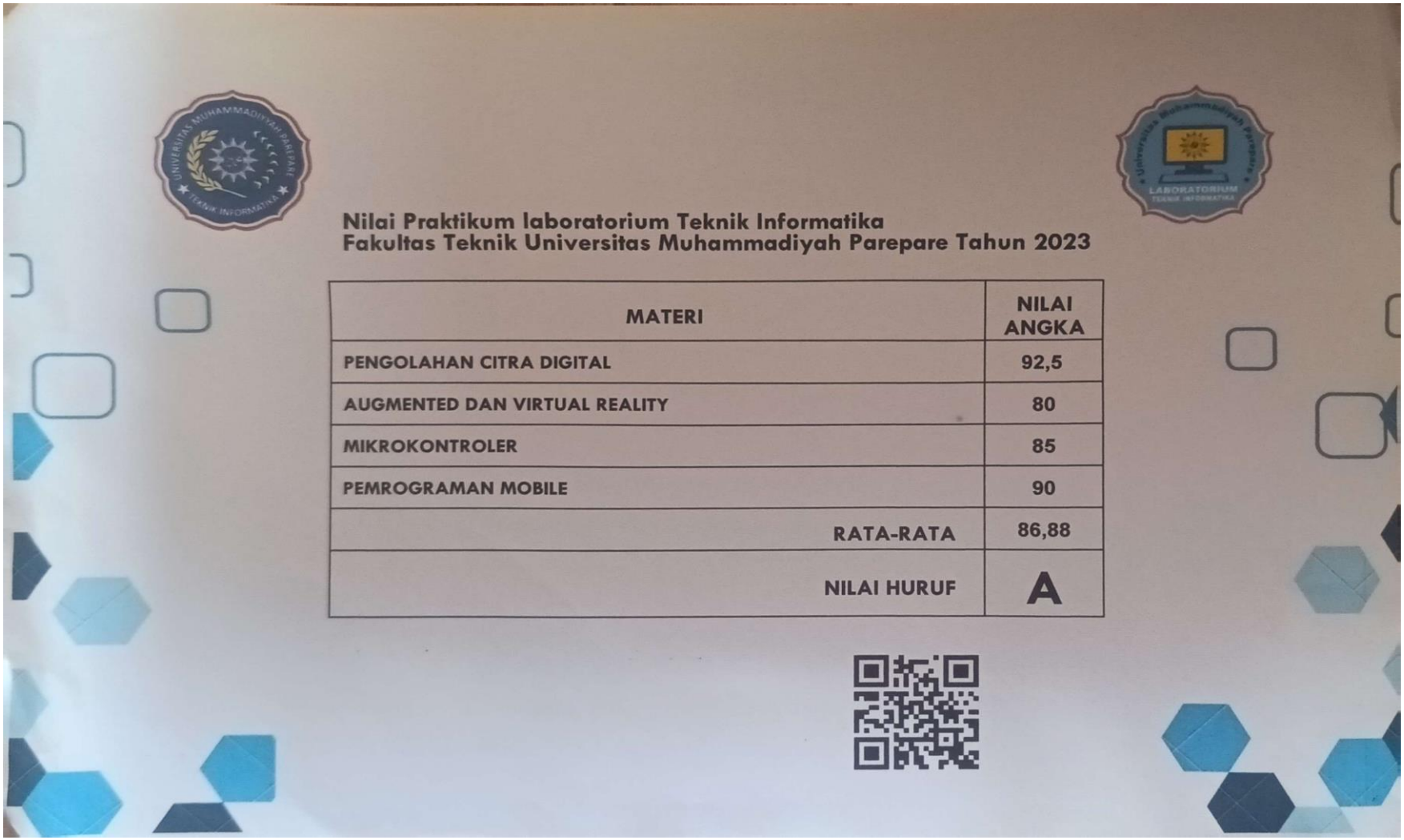
Parepare, 15 Juni 2023

KETUA PRODI
TEKNIK INFORMATIKA

Martina, S.Kom., M.kom
NBM. 1162 680

KEPALA
LABORATORIUM

Ir. Untung Suwardoyo, S.Kom., MT., IPP.
NBM. 1288 973



Nilai Praktikum laboratorium Teknik Informatika
Fakultas Teknik Universitas Muhammadiyah Parepare Tahun 2023

MATERI	NILAI ANGKA
PENGOLAHAN CITRA DIGITAL	92,5
AUGMENTED DAN VIRTUAL REALITY	80
MIKROKONTROLER	85
PEMROGRAMAN MOBILE	90
RATA-RATA	86,88
NILAI HURUF	A