Py_to_PDF

May 8, 2025

[ ]:
```python
#!/usr/bin/env python3
"""
Capture and processing module for water-line detection.

Handles both Raspberry Pi (with Picamera2 and GPIO) and PC modes:
 - Captures images (auto and manual exposure) on Pi,
 - Processes saved images on PC,
 - Posts results to ORC API if available,
 - Provides dummy/simulation modes for testing.
"""
import os
import time
import shutil
import logging
from datetime import datetime

# Attempt to import Pi-specific hardware; fall back if unavailable
try:
    import RPi.GPIO as GPIO
    from picamera2 import Picamera2
    RPI_AVAILABLE = True
except ImportError:
    RPI_AVAILABLE = False
    GPIO = None
    Picamera2 = None

from PIL import Image

# Import processing and ORC utilities
from wd__modules.wd__image_processing_cycle import process_image
from wd__modules.wd__orc_api_cycle import ORC

# Configure logger for this module
logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)
if not logger.hasHandlers():
    ch = logging.StreamHandler()
```

```
    ch.setFormatter(logging.Formatter("%(asctime)s [%(levelname)s]␣
 ↪%(message)s"))
    logger.addHandler(ch)

# Global ORC client placeholder
_GLOBAL_ORC = None


def set_orc_instance(orc):
    """
    Store a global ORC client instance for future uploads.

    Args:
        orc: Initialized ORC client or None
    """
    global _GLOBAL_ORC
    _GLOBAL_ORC = orc


def post_to_orc(image_path: str, waterline_pixel_height: int, site_id: int =␣
 ↪12):
    """
    Send waterline and image data to the ORC API.

    Args:
        image_path: Filepath of the processed overview image
        waterline_pixel_height: Detected waterline row (px), or None
        site_id: ORC site identifier
    """
    # Skip if detection failed or no client
    if waterline_pixel_height is None:
        logger.warning(f"No waterline found for {image_path}. Skipping ORC␣
 ↪upload.")
        return
    if _GLOBAL_ORC is None:
        logger.info("No ORC instance available. Skipping ORC post.")
        return

    # Prepare and post time-series data
    data_timeseries = {
        "timestamp": datetime.utcnow(),
        "site_id": site_id,
        "h": int(waterline_pixel_height)
    }
    try:
        _GLOBAL_ORC.post_timeseries(site_id, data_timeseries)
        logger.info(f"Posted timeseries to ORC: {data_timeseries}")
```

```
        # Post the image via the video endpoint
        data_video = {"timestamp": datetime.utcnow(), "camera_config": 18}
        _GLOBAL_ORC.post_video(data=data_video, file=None, img=image_path)
        logger.info(f"Posted image to ORC video endpoint: {image_path}")
    except Exception as e:
        logger.error(f"Failed posting data to ORC: {e}")


def simulate_burst_and_process_rpi(cycle_interval: float):
    """
    Dummy mode for Raspberry Pi: blink LED and wait, simulating capture.

    Args:
        cycle_interval: Seconds to sleep between simulated captures
    """
    logger.info("Running dummy mode on Raspberry Pi.")
    if not RPI_AVAILABLE:
        logger.warning("simulate_burst_and_process_rpi called but RPi not␣
 ↪available.")
        return
    pinLED = 17
    GPIO.setwarnings(False)
    GPIO.setup(pinLED, GPIO.OUT)
    try:
        for _ in range(2):
            GPIO.output(pinLED, GPIO.HIGH)
            logger.info("Dummy capture: LED ON, simulating algorithm 5s...")
            time.sleep(5)
            GPIO.output(pinLED, GPIO.LOW)
            logger.info("Dummy capture done, resting...")
            time.sleep(cycle_interval)
    finally:
        GPIO.output(pinLED, GPIO.LOW)
        logger.info("Dummy mode finished.")


def simulate_burst_and_process_pc():
    """
    Dummy mode for PC: simple delay to simulate work.
    """
    logger.info("Running dummy mode on PC.")
    time.sleep(5)
    logger.info("Dummy PC mode complete.")


def capture_burst_and_process(
```

```
    raw_folder: str,
    processed_folder: str,
    burst_intervals,
    cycle_interval: float,
    processing_params: dict,
    system: str = "pc",
    dummy_mode: bool = False
):
    """
    Top-level entry: branch to dummy, Pi, or PC routines.

    Args:
        raw_folder: Directory to save raw captures
        processed_folder: Directory to save processed outputs
        burst_intervals: List of delays between burst frames (unused on Pi)
        cycle_interval: Delay between cycles (unused on Pi)
        processing_params: Dict of detection settings
        system: "raspberry_pi" or "pc"
        dummy_mode: If True, run simulation instead of real capture
    """
    if dummy_mode:
        # Choose simulation by system type
        if system.lower() == "raspberry_pi" and RPI_AVAILABLE:
            simulate_burst_and_process_rpi(cycle_interval)
        else:
            simulate_burst_and_process_pc()
        return

    # Ensure output directories exist
    os.makedirs(raw_folder, exist_ok=True)
    os.makedirs(processed_folder, exist_ok=True)

    # Delegate to system-specific implementation
    if system.lower() == "raspberry_pi" and RPI_AVAILABLE:
        capture_burst_and_process_rpi(
            raw_folder,
            processed_folder,
            burst_intervals,
            cycle_interval,
            processing_params
        )
    else:
        capture_burst_and_process_pc(
            raw_folder,
            processed_folder,
            processing_params
        )
```

```python
def capture_burst_and_process_rpi(
    raw_folder: str,
    processed_folder: str,
    burst_intervals,
    cycle_interval: float,
    processing_params: dict
):
    """
    Real capture on Raspberry Pi using Picamera2:
     - Auto-exposure image, optional manual exposure at night,
     - Save to raw_folder, process and post results,
     - Copy to USB if mounted.

    Args:
        raw_folder: Directory for raw images
        processed_folder: Directory for processed outputs
        processing_params: Dict with angle, box_height, min_distance, sigma
    """
    logger.info("Pi mode: starting real capture with auto-exposure.")
    if not RPI_AVAILABLE:
        logger.warning("capture_burst_and_process_rpi called but RPi not␣
 ↪available.")
        return

    # Setup LED and camera
    pinLED = 4
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(pinLED, GPIO.OUT)

    picam2 = Picamera2()
    config = picam2.create_still_configuration(main={"size":␣
 ↪tuple(CONFIG["capture_params"]["resolution"])})
    picam2.configure(config)

    try:
        # Warm up auto-exposure
        picam2.start()
        GPIO.output(pinLED, GPIO.HIGH)
        logger.info("LED ON: adjusting auto-exposure for 5s...")
        time.sleep(5)

        # Capture auto-exposure frame
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S_%f")
        frame = picam2.capture_array()
```

```
        meta = picam2.capture_metadata()
        exp_time = meta.get("ExposureTime")
        suffix = f"autoexp_{exp_time}us" if exp_time else "autoexp"
        filename = f"capture_{timestamp}_{suffix}.jpg"
        img_path = os.path.join(raw_folder, filename)
        Image.fromarray(frame).save(img_path)
        logger.info(f"Captured image: {img_path}")

        # Turn off LED to conserve power
        GPIO.output(pinLED, GPIO.LOW)

        # Copy image to USB if mounted
        usb_path = "/media/bjorn/C0D9-1D92"
        if os.path.ismount(usb_path):
            dest = os.path.join(usb_path, os.path.basename(img_path))
            try:
                shutil.copy2(img_path, dest)
                logger.info(f"Copied to USB: {dest}")
            except Exception as e:
                logger.warning(f"USB copy failed: {e}")
        else:
            logger.error(f"USB not mounted at {usb_path}, skipping copy.")

        # Determine which image to process: manual at night
        hour = datetime.now().hour
        process_path = img_path
        if hour >= 20 or hour < 7:
            logger.info("Night mode: capturing manual exposure image.")
            GPIO.output(pinLED, GPIO.HIGH)
            time.sleep(1)
            picam2.set_controls({"AeEnable": False, "ExposureTime": 150000,␣
↪"AnalogueGain": 1.0})
            time.sleep(1)
            mframe = picam2.capture_array()
            mfilename = f"capture_{datetime.now().
↪strftime('%Y%m%d_%H%M%S_%f')}_manual150000us.jpg"
            mpath = os.path.join(raw_folder, mfilename)
            Image.fromarray(mframe).save(mpath)
            logger.info(f"Captured manual image: {mpath}")
            GPIO.output(pinLED, GPIO.LOW)
            process_path = mpath

        # Process selected image and post results
        result = process_image(
            process_path,
            processed_folder,
            angle=processing_params["angle"],
```

```
            box_height=processing_params["box_height"],
            min_distance=processing_params["min_distance"],
            sigma=processing_params["sigma"],
            system="raspberry_pi"
        )
        if result:
            overview, data = result
            logger.info(f"Detected waterline at px: {data['primary_index']}")
            post_to_orc(overview, data['primary_index'],␣
 ↪site_id=CONFIG['orc']['site_id'])
        else:
            logger.warning(f"No data returned for {process_path}.")

    except Exception as err:
        logger.error(f"Error during capture: {err}", exc_info=True)
    finally:
        # Cleanup: LED off, stop and close camera
        GPIO.output(pinLED, GPIO.LOW)
        try:
            picam2.stop()
        except Exception:
            pass
        try:
            picam2.close()
        except Exception:
            pass
        time.sleep(2)
        logger.info("Capture routine complete.")


def capture_burst_and_process_pc(
    raw_folder: str,
    processed_folder: str,
    processing_params: dict
):
    """
    PC mode: process all existing images in raw_folder in batch.

    Args:
        raw_folder: Directory containing input images
        processed_folder: Directory for saving outputs
        processing_params: Dict with detection settings
    """
    logger.info("PC mode: processing existing images.")
    images = [os.path.join(raw_folder, f) for f in os.listdir(raw_folder)
              if f.lower().endswith((".jpg", ".jpeg", ".png"))]
    if not images:
```

```
        logger.info("No images found, exiting PC processing.")
        return

    site_id = CONFIG["orc"]["site_id"]
    for idx, img in enumerate(images, 1):
        logger.info(f"Processing [{idx}/{len(images)}]: {img}")
        try:
            result = process_image(
                img,
                processed_folder,
                angle=processing_params["angle"],
                box_height=processing_params["box_height"],
                min_distance=processing_params["min_distance"],
                sigma=processing_params["sigma"],
                system="pc"
            )
            if result:
                overview, data = result
                post_to_orc(overview, data['primary_index'], site_id=site_id)
        except Exception as e:
            logger.error(f"Error processing {img}: {e}", exc_info=True)
    logger.info("PC batch processing complete.")


if __name__ == "__main__":
    # Example standalone invocation for testing
    raw_folder = "/home/bjorn/Desktop/wd__directory/wd__Calibration/
 ↪Shutterspeed_test"
    os.makedirs(raw_folder, exist_ok=True)
    capture_burst_and_process(
        raw_folder,
        raw_folder,
        burst_intervals=[0],
        cycle_interval=10,
        processing_params={"angle":0, "box_height":100, "min_distance":50,␣
 ↪"sigma":1.5},
        system="raspberry_pi",
        dummy_mode=False
    )
```