International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 11 , November - 2025

Automotive Glare Suppression and Night Vision Assistance

Salava V Satyanarayana
Electrical and Electronics Engineering Hyderabad Institute of Technology and Management Hyderabad, India

D Nikesh
Electrical and Electronics Engineering Hyderabad Institute of Technology and Management Hyderabad, India

B Aravind
Electrical and Electronics Engineering Hyderabad Institute of Technology and Management Hyderabad, India

B Ganesh
Electrical and Electronics Engineering Hyderabad Institute of Technology and Management Hyderabad, India

K Manasa
Electrical and Electronics Engineering Hyderabad Institute of Technology and Management Hyderabad, India

***Abstract* - The project “Automotive Glare Suppression and Night Vision Assistance” aims to enhance driver visibility and safety during nighttime driving by integrating image processing and embedded vision technology. A Raspberry Pi 3 B+ serves as the core processing unit, interfaced with a Pi camera to continuously capture real-time video from the vehicle’s front view. The system processes these images to detect and suppress high-intensity glare caused by oncoming vehicle headlights using adaptive filtering. Simultaneously, the setup assists the driver in low-light conditions by improving image brightness and clarity through night-vision enhancement techniques. The processed output is displayed on a 7-inch HDMI screen for live monitoring. An SD card is used for data storage, while a regulated power supply ensures reliable system operation. This cost-effective vision-based assistance system improves driving comfort and situational awareness, reducing accidents caused by temporary blindness**

***Keywords* - Glare reduction, Night vision, Driver assistance system, Embedded imaging, Raspberry Pi, Real-time video, Low-light processing, Vehicle safety, Adaptive filtering, Smart vision system.**

I. INTRODUCTION

Driving at night poses several challenges due to reduced visibility and the intense glare produced by the headlights of oncoming vehicles.[1] These sudden bursts of bright light can temporarily blind drivers, leading to delayed reaction times and, in severe cases, road accidents. To address this critical issue, the project “Automotive Glare Suppression and Night Vision Assistance” has been developed as an intelligent, low-cost driver assistance system aimed

at improving safety and comfort during nighttime driving. The system uses a Raspberry Pi 3 B+ microcomputer as the central controller, connected to a Pi camera that continuously captures live video from the front of the vehicle. The captured video is displayed on a 7-inch HDMI screen, allowing the driver to have a clearer and glare-free view of the road.[2] The Pi camera automatically adjusts brightness and contrast levels depending on the lighting conditions, helping to reduce glare and improve visibility in dark or low-light environments.

A power supply unit ensures reliable and continuous operation, while an SD card. A power supply unit ensures reliable and continuous operation, while an SD card provides the necessary storage for the operating system and captured data. The system’s compact design allows it to be easily integrated into existing vehicles without major modifications. By providing better visibility during night driving, this project helps reduce accidents caused by headlight glare and poor illumination. Overall, the Automotive Glare Suppression and Night Vision Assistance System demonstrate how embedded vision technology can be effectively applied to real-world automotive safety. It enhances driver awareness, minimizes fatigue, and ensures safer driving conditions — especially on highways and poorly lit roads.

II. HARDWARE SPECIFICATIONS

A.RASPBERRY PI 3 B+

The Raspberry Pi 3 Model B+ acts as the heart of the system. It performs all control and image-processing tasks. The processor used is a Broadcom BCM2837B0, 64-bit ARM Cortex-A53 quad-core CPU running at 1.4GHz, with 1GB LPDDR2 RAM. [5]It provides multiple connectivity options, including four USB ports, HDMI, CSI for camera input, DSI for display output, and 40 GPIO pins for peripheral interfacing.

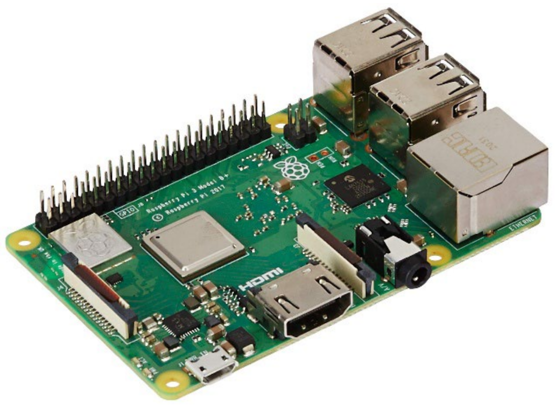

Fig 2.1. Raspberry Pi 3

B.PI CAMERA

The Pi Camera Module is used to capture real-time video of the road ahead. It connects to the Raspberry Pi via the CSI ribbon cable. The camera has a 5MP resolution and supports 1080p30, 720p60, and VGA90 video modes. It provides sharp image capture even in low-light conditions and serves as the key sensing component of the system.

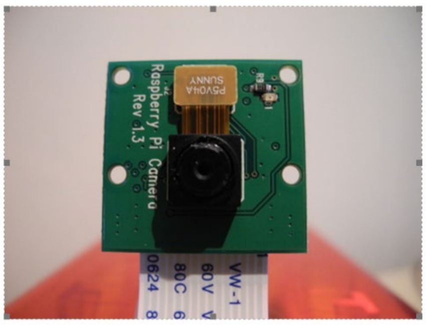

Fig 2.2. Pi Camera

C.SD CARD

An SD card (Secure Digital card) is a small, portable storage device used to store and transfer data such as images, videos, software, and operating systems. The 16GB SD card is a key part of the Raspberry Pi; it provides the initial storage for the Operating System and files.

D.7 INCH HDMI SCREEN

A 7-inch HDMI LCD screen is used to display the processed video output in real time. The display is connected to the HDMI port of the Raspberry Pi and operates on a 5V DC supply. It provides a resolution of 1024x600 pixels and can optionally support touch input through a USB interface.

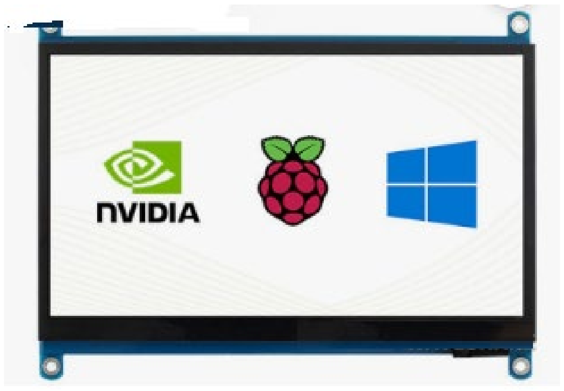

Fig 2.3. 7 Inch Hdmi Screen

III. WORKING

The Automotive Glare Suppression and Night Vision Assistance system works by capturing the road view through a camera mounted at the front of the vehicle.[3]This live video is continuously sent to the Raspberry Pi, which acts as the main processing unit. As the frames are received, the system analyzes the brightness levels to identify areas where strong headlight glare from oncoming vehicles is present. Once these bright regions are detected, the software reduces their intensity so that the glare does not affect the driver's vision. At the same time, the system enhances darker areas of the frame to improve overall visibility during night-time conditions. After processing, the improved and glare-free video is displayed on a 7-inch HDMI screen in real time, allowing the driver to clearly see the road ahead.[4] An LED indicator remains ON during operation to show that the system is actively processing video. This entire cycle of capturing, analyzing, suppressing glare, enhancing visibility, and displaying output repeats continuously, providing the driver with a safer and more comfortable night-driving experience.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 11 , November - 2025

A.BLOCK DIAGRAM

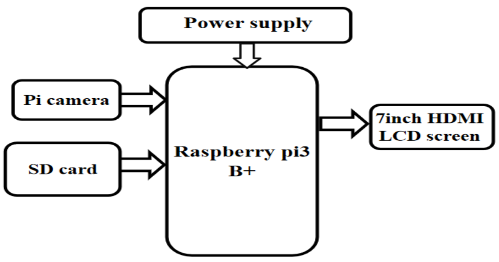

Fig 3.1. A block diagram illustrating the working of Automotive glare suppression and night vision assistance

B.CODE IMPLEMENTATION

```
import os
import time
import threading
from concurrent.futures import
ThreadPoolExecutor
from functools import partial
import numpy as np
import cv2
from picamera2 import Picamera2
import pygame
from flask import Flask, Response
# Try to import TFLite runtime (preferred) or full TF
interpreter
TFLITE_AVAILABLE = False
interpreter = None
try:
    from tflite_runtime.interpreter import Interpreter
    TFLITE_AVAILABLE = True
except Exception:
    try:
        from tensorflow.lite.python.interpreter import
Interpreter  # type: ignore
        TFLITE_AVAILABLE = True
    except Exception:
        TFLITE_AVAILABLE = False
# ----------------- Config -----------------
DISPLAY_W, DISPLAY_H = 1024, 600
CAPTURE_W, CAPTURE_H = 640, 360   # processing
resolution (lower for Pi)
FPS = 15
JPEG_QUALITY = 70
TFLITE_MODEL_PATH = "glare_detector.tflite"  # if
exists, will be used
INFERENCE_THREADS = 1
BRIGHT_PIXEL_THRESHOLD = 0.005  # heuristic bright
ratio threshold
EMA_ALPHA = 0.25  # smoothing for mask & output
# Flask app
app = Flask(__name__)
# Shared data
shared_frame = None
shared_frame_lock = threading.Lock()
running = True
# Threadpool for ML inference
executor =
ThreadPoolExecutor(max_workers=INFERENCE_THREA
DS)
# Optional tflite interpreter wrapper
def load_tflite_model(path):
    global interpreter
    if not TFLITE_AVAILABLE:
        print("[ML] TFLite runtime not available. Falling back
to heuristic.")
        return None
    if not os.path.isfile(path):
        print(f"[ML] Model file not found at '{path}'. Falling
back to heuristic.")
        return None
    try:
        interp = Interpreter(model_path=path)
        interp.allocate_tensors()
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 11 , November - 2025

```
        # get input/output details
        input_details = interp.get_input_details()
        output_details = interp.get_output_details()
        print(f"[ML] Loaded TFLite model {path}")
        return (interp, input_details, output_details)
    except Exception as e:
        print("[ML] Failed to load tflite model:", e)
        return None
tflite_model = load_tflite_model(TFLITE_MODEL_PATH)
# ----------------- ML Inference / Heuristic -----------------
def predict_glare_mask_tflite(model_tuple, frame):
    """
    Runs model inference on `frame` and returns a mask
(float32 [0..1], same w,h as frame).
    Expects model input size and preprocess accordingly.
    """
    try:
        interp, input_details, output_details = model_tuple
        h_in, w_in = input_details[0]['shape'][1:3]
        # Resize RGB input and normalize to [0,1]
        inp = cv2.resize(frame, (w_in, h_in))
        # Model input may expect float32 or uint8
        if input_details[0]['dtype'] == np.float32:
            inp = inp.astype(np.float32) / 255.0
        else:
            inp = inp.astype(input_details[0]['dtype'])
        inp = np.expand_dims(inp, axis=0)
        interp.set_tensor(input_details[0]['index'], inp)
        interp.invoke()
        out = interp.get_tensor(output_details[0]['index'])
        # Attempt to convert to single-channel mask
        if out.ndim == 4:
            # often shape (1, h, w, 1) or (1, h, w, c)
            mask = out[0, :, :, 0] if out.shape[-1] >= 1 else out[0,
:, :, 0]
        elif out.ndim == 3:
            mask = out[0, :, :]
        else:
            # fallback flatten
            mask = out.squeeze()
        # normalize to 0..1
        mask = mask.astype(np.float32)
        mask = (mask - mask.min()) / (mask.max() -
mask.min() + 1e-8)
        # resize to frame size
        mask = cv2.resize(mask, (frame.shape[1],
frame.shape[0]))
        return mask
    except Exception as e:
        print("[ML] Inference error:", e)
        return None
def heuristic_glare_mask(frame):
    """Simple fast heuristic to detect bright regions without
ML."""
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    v = hsv[:, :, 2]
    # bright pixels > 240
    mask = (v > 240).astype(np.float32)
    # morphological clean and small gaussian blur to smooth
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN,
np.ones((5,5), np.uint8))
    mask = cv2.GaussianBlur(mask, (11,11), 0)
    # normalize
    if mask.max() > 0:
        mask = mask / mask.max()
    return mask
# ----------------- Tone mapping & blending -----------------
def local_tone_map(frame, mask):
    """
    Apply local tone mapping controlled by mask (0..1).
    - compress highlights where mask ~1
    - enhance shadows where mask ~0 but global scene dark
    Returns result BGR 8-bit.
    """
    # Convert to HSV and operate on V
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV).astype(np.float32)
    h, s, v = cv2.split(hsv)
    # Smooth mask and clip
    mask_s = cv2.GaussianBlur(mask, (21,21), 0)
    mask_s = np.clip(mask_s, 0.0, 1.0)
    # Compute local compression: where mask high -> compress v
    # Use soft compression: v' = a + (v - a) * factor, factor <1 on bright regions
    # a is threshold base (~220), convert to 0-255
    a = 220.0
    # factor map: between 0.6 (strong compression) and 1.0 (no compression)
    factor_map = 1.0 - 0.4 * mask_s  # 1.0 to 0.6
    v_comp = a + (v - a) * factor_map

    # Boost shadows a bit using CLAHE on downsampled V to save CPU
    v_u8 = np.clip(v_comp, 0, 255).astype(np.uint8)
    v_small = cv2.resize(v_u8, (v_u8.shape[1]//2, v_u8.shape[0]//2))
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    v_small = clahe.apply(v_small)
    v_boost = cv2.resize(v_small, (v_u8.shape[1], v_u8.shape[0]))
    # blend CLAHE boosted with compressed using a midtone weight (preserve highlights)
    mid_weight = 0.35
    v_final = (1-mid_weight) * v_comp + mid_weight * v_boost.astype(np.float32)
    # Optional slight gamma correction to brighten midtones
    gamma = 1.05
    v_final = 255.0 * ((v_final/255.0)**(1.0/gamma))
    # clip and convert back
    v_final = np.clip(v_final, 0, 255).astype(np.uint8)
    hsv_out = cv2.merge([h.astype(np.uint8), s.astype(np.uint8), v_final])
    out = cv2.cvtColor(hsv_out, cv2.COLOR_HSV2BGR)
    return out
# ----------------- ML orchestrator -----------------
def infer_mask_async(frame):
    """
    Submit inference task to threadpool. Returns Future pointing to mask (or None).
    """
    if tflite_model:
        return executor.submit(predict_glare_mask_tflite, tflite_model, frame)
    else:
        # return immediate heuristic (wrap into future-like by using executor)
        return executor.submit(heuristic_glare_mask, frame)
# ----------------- Capture & Process Loop -----------------
ema_mask = None  # exponential moving average for mask
ema_frame = None  # optional EMA on output for smoothing
def capture_process_loop():
    """Capture frames, optionally run ML inference, apply local tone-mapping when needed,
       and update shared_frame for display/streaming."""
    global shared_frame, running, ema_mask, ema_frame
    # Create pygame surface later in display thread; here only process frames
    pending_future = None
    pending_frame_for_infer = None
    while running:
        frame = picam2.capture_array()  # BGR 640x360
        # Step 1: decide whether to run ML/heuristic detection
# We run a fast heuristic to decide if heavy inference is needed:
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
avg_brightness = gray.mean()
# If very bright average (night scenes with headlights), always run detection
need_detection = False
# quick bright-ratio heuristic (cheap)
bright_pixels_quick = np.sum(gray > 240)
if bright_pixels_quick / gray.size > 0.002 or avg_brightness > 160:
    need_detection = True

# If we don't need detection, use heuristic mask cheap (fast)
if not need_detection and not tflite_model:
    mask = heuristic_glare_mask(frame)
else:
    # submit inference if not already pending
    if pending_future is None:
        pending_frame_for_infer = cv2.resize(frame, (320, 180))  # smaller for model speed
        pending_future = infer_mask_async(pending_frame_for_infer)
        # keep processing current frame with heuristic until ML result returns
        mask = heuristic_glare_mask(frame)
    else:
        # check if future done
        if pending_future.done():
            try:
                result_mask = pending_future.result(timeout=0.01)
                pending_future = None
                pending_frame_for_infer = None
                # if result_mask is None fallback to heuristic
                if result_mask is None:
                    mask = heuristic_glare_mask(frame)
                else:
                    # result_mask may be smaller; ensure same size
                    if result_mask.shape != (frame.shape[0], frame.shape[1]):
                        result_mask = cv2.resize(result_mask, (frame.shape[1], frame.shape[0]))
                    mask = result_mask.astype(np.float32)
            except Exception:
                # still not ready or error
                mask = heuristic_glare_mask(frame)
        else:
            # still waiting: use heuristic
            mask = heuristic_glare_mask(frame)
# Step 2: conditional apply: only if sufficient bright-region ratio
bright_ratio = np.sum(mask > 0.6) / mask.size
apply_filter = bright_ratio > BRIGHT_PIXEL_THRESHOLD
# Smooth mask via EMA
if ema_mask is None:
    ema_mask = mask
else:
    ema_mask = (1 - EMA_ALPHA) * ema_mask + EMA_ALPHA * mask

# If apply_filter True use ML-guided mapping; else show original frame
if apply_filter:
    out = local_tone_map(frame, ema_mask)
else:
    out = frame
# Optionally smooth output slightly (temporal)
if ema_frame is None:
    ema_frame = out.astype(np.float32)
else:
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 11 , November - 2025

```
            ema_frame = (1 - EMA_ALPHA) * ema_frame +
EMA_ALPHA * out.astype(np.float32)
        out_display = np.clip(ema_frame, 0,
255).astype(np.uint8)
        # Place resulting frame into shared variable
        with shared_frame_lock:
            shared_frame = out_display.copy()
        # small sleep to yield CPU
        time.sleep(1.0 / FPS * 0.2)
# ---------------- HDMI Display Thread (pygame, headless) -
----------------
def display_thread():
    global shared_frame, running
    pygame.init()
    screen = pygame.display.set_mode((DISPLAY_W,
DISPLAY_H), pygame.FULLSCREEN)
    pygame.display.set_caption("AI Anti-Glare Live Feed")
    clock = pygame.time.Clock()
    while running:
        with shared_frame_lock:
            frame = None if shared_frame is None else
shared_frame.copy()
        if frame is not None:
            # convert BGR -> RGB for pygame
            rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            surf = pygame.surfarray.make_surface(np.rot90(rgb))
            surf = pygame.transform.scale(surf, (DISPLAY_W,
DISPLAY_H))
            screen.blit(surf, (0,0))
            pygame.display.flip()
        clock.tick(FPS)
        # Quit logic by keyboard (q)
        for ev in pygame.event.get():
            if ev.type == pygame.KEYDOWN and ev.key ==
pygame.K_q:
                print("[MAIN] Quit key pressed.")
                stop()
            elif ev.type == pygame.QUIT:
                stop()
    pygame.quit()
# ---------------- Flask streaming ----------------
HTML = """
<!doctype html>
<title>AI Anti-Glare Stream</title>
<body style="background:black;text-
align:center;color:#0ff;">
<h1>AI Anti-Glare Live Stream</h1>
<img src="/video_feed" style="width:90%;">
</body>
"""
@app.route('/')
def index():
    return HTML
def mjpeg_generator():
    while running:
        with shared_frame_lock:
            frame = None if shared_frame is None else
shared_frame.copy()
        if frame is None:
            continue
        # encode JPEG
        ret, buf = cv2.imencode('.jpg', frame,
[int(cv2.IMWRITE_JPEG_QUALITY), JPEG_QUALITY])
        if not ret:
            continue
        jpg = buf.tobytes()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + jpg + b'\r\n')
        time.sleep(1.0 / max(FPS,5))
@app.route('/video_feed')
def video_feed():
    return Response(mjpeg_generator(),
mimetype='multipart/x-mixed-replace; boundary=frame')
```

```
# ----------------- Utilities & Shutdown -----------------

def stop():

    global running

    print("[MAIN] Stopping...")

    running = False

# ----------------- Main entry -----------------

if __name__ == "__main__":

    print("[MAIN] Starting advanced AI HDR stream")

    # Picamera2 init

    picam2 = Picamera2()

    picam2.preview_configuration.main.size =
(CAPTURE_W, CAPTURE_H)

    picam2.preview_configuration.main.format = "RGB888"

  picam2.preview_configuration.controls.FrameRate = FPS

    picam2.configure("preview")

    picam2.start()

    print(f"[CAM] capture
{CAPTURE_W}x{CAPTURE_H} @ {FPS}FPS")

    # Start threads

    t_capture =
threading.Thread(target=capture_process_loop,
daemon=True)

    t_display = threading.Thread(target=display_thread,
daemon=True)

    t_capture.start()

    t_display.start()

    try:

        app.run(host='0.0.0.0', port=8000, threaded=True)

    except KeyboardInterrupt:

        stop()

    # graceful stop

    t_capture.join(timeout=1.0)

    t_display.join(timeout=1.0)

    print("[MAIN] Exiting")
```

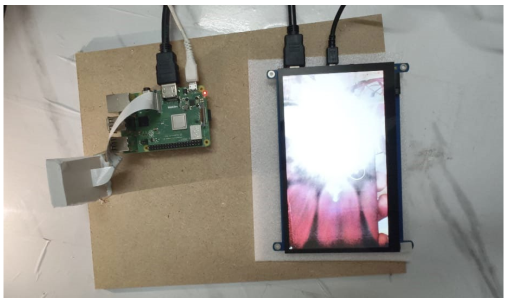

Fig 3.2 Proto Type of Automotive Glare Suppression and Night Vision Assistance

IV. CONCLUSION

The proposed system demonstrates an effective and low-cost solution for enhancing night driving safety. By using Raspberry Pi and a Pi camera, the setup provides glare reduction and better visibility in dark conditions. The live video output on the HDMI screen assists drivers in identifying obstacles and maintaining focus on the road. This project showcases how embedded systems can be applied in real-world automotive safety enhancements.

A.FUTURE SCOPE

In the future, this system can be further developed to achieve complete automation of headlight control in vehicles. By integrating the glare suppression module with the vehicle's lighting circuit, the system can automatically dim or adjust the intensity of headlights whenever glare from oncoming vehicles is detected. This would help prevent glare at its source rather than only at the visual output, thereby enhancing safety for both drivers.

REFERANCE

[1] Raspberry Pi Foundation, "Raspberry Pi 3 Model B+ Technical Specifications," 2020.

[2] S. Ullah et al., "Single-equipment with multiple-application for an automated robot-car control system," Sensors, vol. 19, no. 662, 2019

[3] Arduino and Raspberry Pi Community Documentation, 2021.

[4] N. K. Patel, "Vision-Based Driver Assistance System for Nighttime Driving," International Journal of Advanced Research in Electronics and Communication Engineering, 2022.

[5] Official Raspberry Pi Documentation, www.raspberrypi.org.

[6] S.Mendis,S.E.Kemeny,andE.R.Fossum,"CMOSactivepixelimage sensor," IEEE Trans. Electron Devices, vol. 41, no. 3, pp. 452–453, Mar. 1994.

[7] S. K. Mendis, S. E. Kemeny, R. C. Gee, B. Pain, C. O. Staller, Q. Kim, and E. R. Fossum, "CMOS active pixel image sensors for highly inte grated imaging systems," IEEE J. Solid-State Circuits, vol. 32, no. 2, pp. 187–196, Feb. 1997. [8] G. P. Weckler, "Operation of p-n junction photodetectors in a photon f lux integrating mode," IEEE J. Solid-State Circuits, vol. SC-2, no. 3, pp. 65–73, Sep. 1967.

[8] B. Dierickx, D. Scheffer, G. Meynants, W. Ogiers, and J. Vlummens, "Random addressable active pixel image sensors," Proc. SPIE—Int. Soc. Opt. Eng., vol. 2950, pp. 2–7, 1996.

[9] D. X. D. Yang and A. El Gamal, "Comparative analysis of SNR for image sensors with enhanced dynamic range," Proc. SPIE—Int. Soc. Opt. Eng., vol. 3649, pp. 197–211, 1999.