Zeeman Effect Analyzer

Slides made and presented by - Abdallah W. Mahmoud 900221058
- Yousif Shaaban 900232840

Overview

Theoretical Background

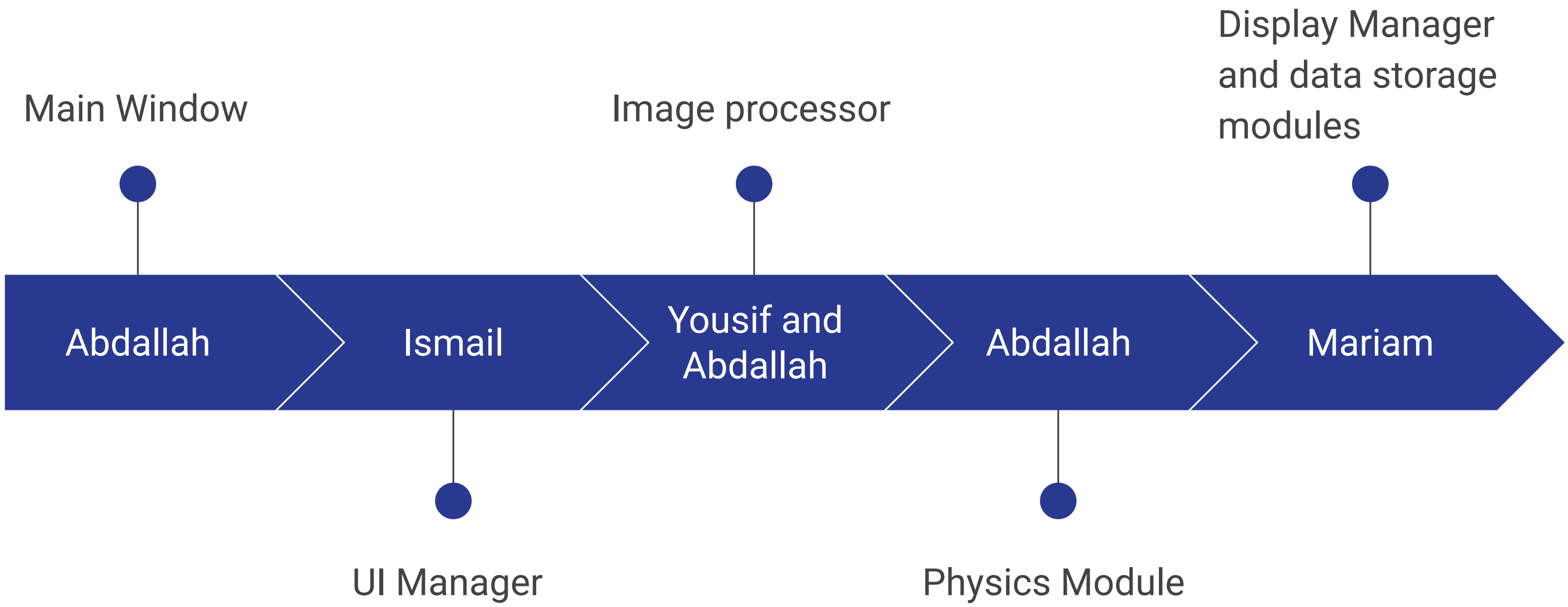
- Theory behind the Zeeman effect
- Description and significance of the Bohr Magneton and specific charge

Modules Description

- Description of all the modules of the program
- Significance and functionality of each module

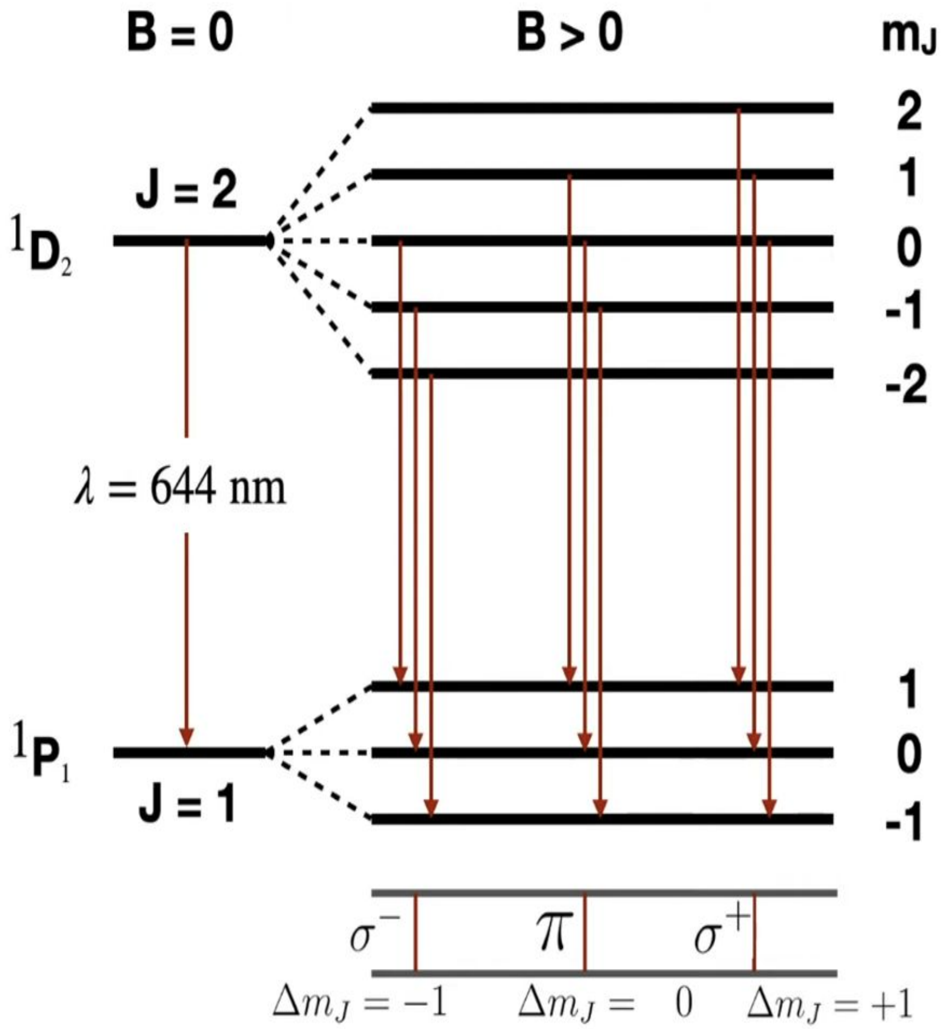
Usage

- Simulation of the code on a sample data
- Verification of the program’s results with theoretical results


Main Window
Abdallah
Ismail
UI Manager
Image processor
Yousif and Abdallah
Abdallah
Physics Module
Display Manager and data storage modules
Mariam

Theoretical Background

In this section, we explain the Zeeman effect in detail



- The splitting of atomic energy levels, and therefore the spectral lines, of an atom when subjected to an external magnetic field.
- The Zeeman effect provided proof of the existence of the electron's magnetic moment, given by μ_J= μ_s+μ_L

$$\mu_L = -\left(\frac{e}{2m_e}\right)L = -\mu_B\left(\frac{L}{\hbar}\right)$$

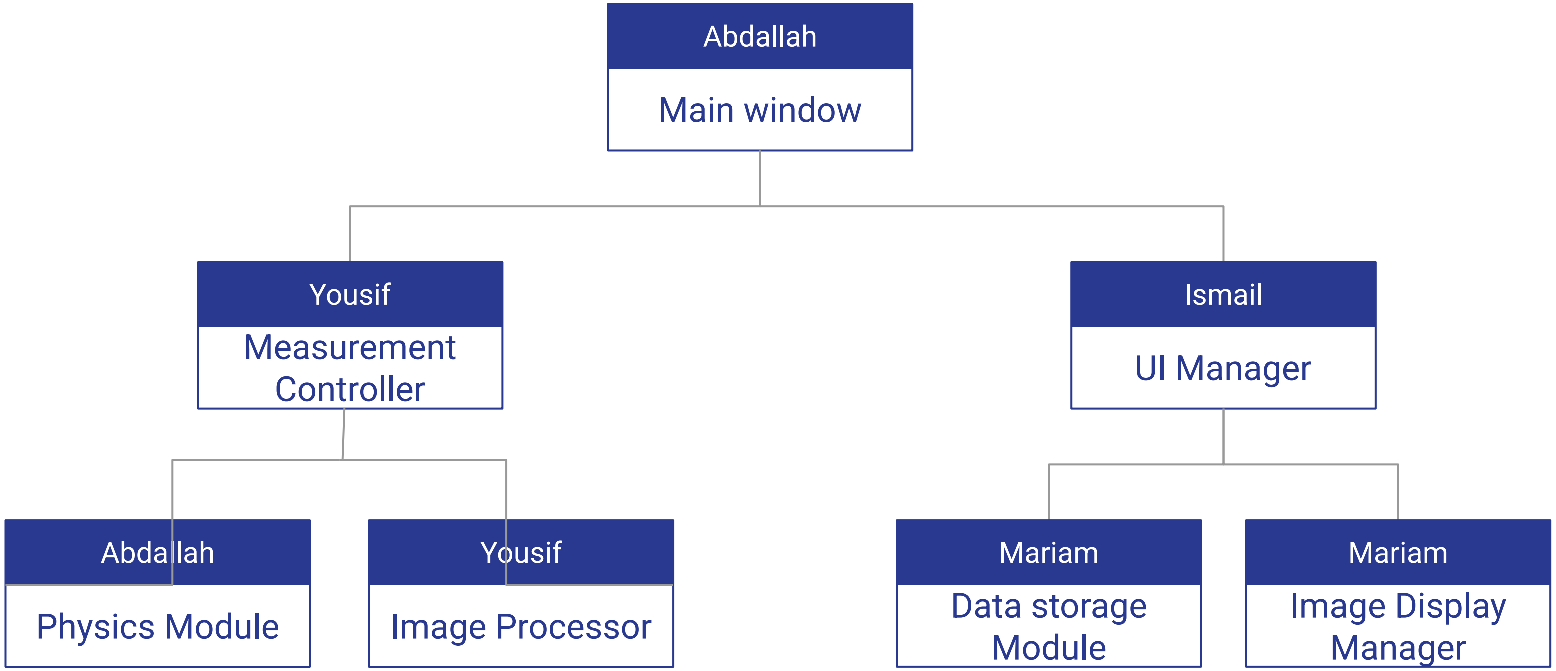
$$\mu_S = -g_S\left(\frac{e}{2m_e}\right)S = -g_S\mu_B\left(\frac{S}{\hbar}\right)$$

$$\Delta E = -\mu_{\{J,z\}}B \qquad \Delta E = g_J\mu_B B\, M_J$$

$$\mu_{\{J,z\}} = -g_J\mu_B\left(\frac{J_z}{\hbar}\right) = -g_J\mu_B M_J$$

Software Architecture

Flowchart


Abdallah
Main window
Yousif
Measurement Controller
Ismail
UI Manager
Abdallah
Physics Module
Yousif
Image Processor
Mariam
Data storage Module
Mariam
Image Display Manager

Main Window

```python
def __init__(self):
    super().__init__()

    # Initialize state variables
    self.images = []
    self.current_image_index = -1
    self.calibration_distance_mm = 2.0
    self.measurements = []

    # Initialize components
    self.ui_manager = UIManager(self)
    self.ui_manager.setup_layout()
    self.image_processor = ImageProcessor()
    self.image_display_manager = ImageDisplayManager(self.image_display, self, self.ui_manager)
    self.measurement_controller = MeasurementController(self, self.ui_manager)
```

Measurement Control

```python
def set_mode(self, mode: Optional[str]):
    intended_mode = mode

    if intended_mode == 'center':
        self.initialize_for_new_measurement()
        self.current_mode = 'center'
    elif intended_mode and intended_mode.startswith('auto_'):
        if self.current_measurement.get('center') is None:
            QMessageBox.information(self.mw, 'Set Center First',
                                    'Please set the center point before defining an annulus for auto-detection.')
            self.current_mode = None
            self.is_defining_annulus = False
            self.auto_detect_limits = {'lower': None, 'upper': None}
            self.mw.update_display()
            return
        else:
            self.current_mode = intended_mode
            self.is_defining_annulus = True
            self.auto_detect_limits = {'lower': None, 'upper': None}
    else:
        self.current_mode = intended_mode
        self.is_defining_annulus = False
        self.auto_detect_limits = {'lower': None, 'upper': None}


    self.mw.update_display()
```

Image Processor

```python
def enhance_image(self):
    if self.image is None:
        raise ValueError("No image loaded.")

    gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    self.processed_image = clahe.apply(blurred)

    return self.processed_image

def auto_detect_radius_refined(self, processed_image, initial_center_x, initial_center_y,
                               radius_lower_limit, radius_upper_limit, center_search_window_half_size=5):
    # Create candidate centers around initial position
    candidate_centers = []
    for dx in range(-center_search_window_half_size*2, center_search_window_half_size*2 + 1, 2):
        for dy in range(-center_search_window_half_size*2, center_search_window_half_size*2 + 1, 2):
            candidate_centers.append((initial_center_x + dx, initial_center_y + dy))


```

Physics Module

```python
@dataclass
class ZeemanMeasurement:
    B_field: float  # Magnetic field strength in Tesla

    wavelength: float  # Central wavelength in meters
    R_center: Optional[float] = None  # Radius of central line in mm
    R_inner: Optional[float] = None   # Radius of inner line in mm
    R_outer: Optional[float] = None   # Radius of outer line in mm

    # [Additional fields omitted for brevity]

def calculate_wavelength_shift(beta: float, beta_c: float, wavelength: float) -> float:
    """Calculate the wavelength shift based on the refracted angles."""
    return wavelength * (np.cos(beta_c) / np.cos(beta) - 1)

def calculate_energy_shift(delta_lambda: float, wavelength: float) -> float:
    """Calculate the energy shift based on the wavelength shift."""
    return PLANCK * LIGHT_SPEED * delta_lambda / (wavelength ** 2)
```

UI Manager

```
def setup_layout(self):
    main_layout = QVBoxLayout(self.mw.centralWidget())

    # Set up navigation
    nav_layout = QHBoxLayout()
    self.mw.prev_image_btn = QPushButton("Previous Image")
    self.mw.prev_image_btn.clicked.connect(self.mw.prev_image)
    self.mw.next_image_btn = QPushButton("Next Image")
    self.mw.next_image_btn.clicked.connect(self.mw.next_image)

    # Set up content area with image display and control panel
    content_layout = QHBoxLayout()
    image_scroll = QScrollArea()
    image_scroll.setWidgetResizable(True)
    # [Layout setup continues...]
```

Image Display Manager

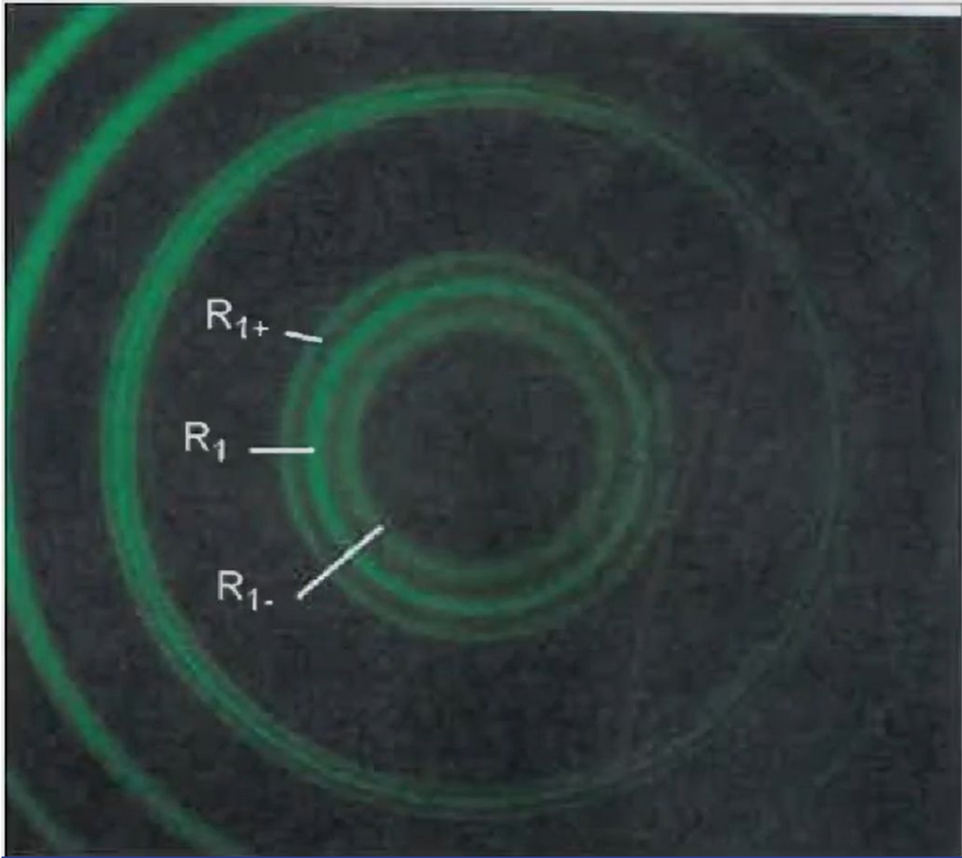
```python
def redraw_image_with_overlays(self):
    if self.mw.current_image_index < 0 or not self.mw.images:
        return

    image_data = self.mw.images[self.mw.current_image_index]
    image = image_data['image'].copy()

    # Convert to OpenCV format for drawing
    display_cv_img = image.copy()

    # Draw calibration points, center, rings, etc.
    mc = self.mw.measurement_controller
```

Data Storage Module

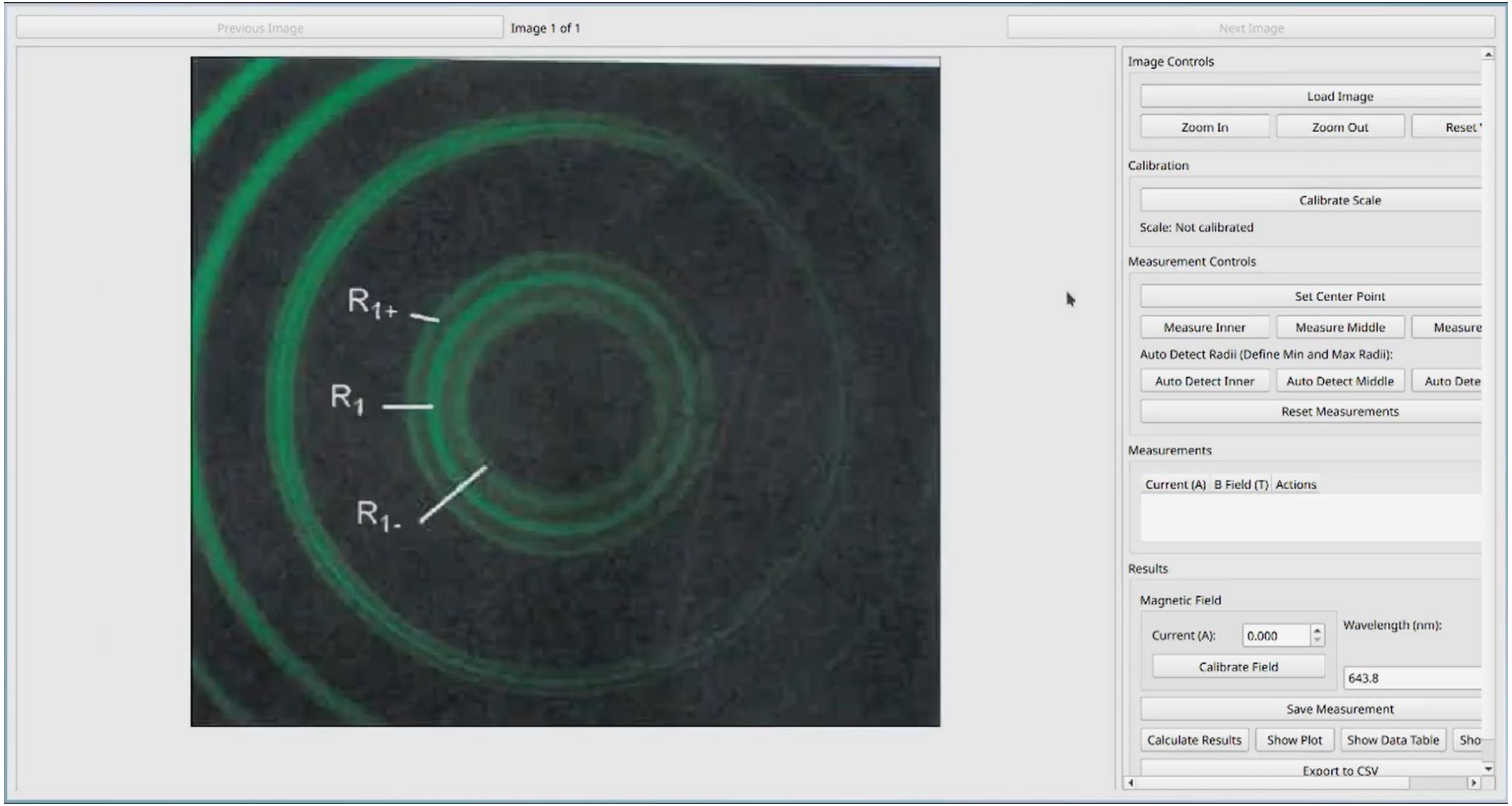
```python
def export_to_csv(self):
    if not self.measurements:
        QMessageBox.warning(self, 'Warning', 'No measurements to export')
        return

    file_path, _ = QFileDialog.getSaveFileName(
        self,
        'Save Measurements',
        '',
        'CSV Files (*.csv)'
    )

    if not file_path:
        return

    L = 150
    n = 1.46

    with open(file_path, 'w', newline='') as f:
        writer = csv.writer(f, delimiter='\t')

        writer.writerow([
            'I(A)', 'B(G)',
            'R_i(mm)', 'R_c(mm)', 'R_o(mm)',
            'α_i(deg)', 'α_c(deg)', 'α_o(deg)',
            'β_i(deg)', 'β_c(deg)', 'β_o(deg)',
            'Δλ_i(nm)', 'Δλ_o(nm)',
            'ΔE_i(eV)', 'ΔE_o(eV)'
        ])

        for m in self.measurements:
            B_field = m.B_field * 1e4

            try:
                slope, intercept = self.calibration_window.calibration_params
                current = (B_field - intercept) / slope
            except (AttributeError, TypeError):
                QMessageBox.warning(self, 'Warning', 'Please calibrate the magnetic field first')
                return

            def calc_angles(r_mm):
                if r_mm is None:
                    return None, None
                alpha = np.arctan(r_mm / L)
                beta = np.arcsin(np.sin(alpha) / n)
                return alpha, beta

            alpha_i, beta_i = calc_angles(m.R_inner)
            alpha_c, beta_c = calc_angles(m.R_center)
            alpha_o, beta_o = calc_angles(m.R_outer)

            def format_val(val, precision=6):
                return f"{val:.{precision}f}" if val is not None else ""

            row = [
                format_val(current, 2),
                format_val(B_field, 2),
                format_val(m.R_inner, 3) if m.R_inner else "",
                format_val(m.R_center, 3) if m.R_center else "",
                format_val(m.R_outer, 3) if m.R_outer else "",
                format_val(np.degrees(alpha_i), 4) if alpha_i is not None else "",
                format_val(np.degrees(alpha_c), 4) if alpha_c is not None else "",
                format_val(np.degrees(alpha_o), 4) if alpha_o is not None else "",
                format_val(np.degrees(beta_i), 4) if beta_i is not None else "",
                format_val(np.degrees(beta_c), 4) if beta_c is not None else "",
                format_val(np.degrees(beta_o), 4) if beta_o is not None else "",
                format_val(m.delta_lambda_i * 1e9, 3) if m.delta_lambda_i else "",  # nm
                format_val(m.delta_lambda_o * 1e9, 3) if m.delta_lambda_o else "",  # nm
                format_val(m.delta_E_i / 1.602176634e-19, 6) if m.delta_E_i else "",  # eV
                format_val(m.delta_E_o / 1.602176634e-19, 6) if m.delta_E_o else ""   # eV
            ]
            writer.writerow(row)

    QMessageBox.information(self, 'Success', f'Measurements exported to {file_path}')
```

Usage

In this section, we compare the program's results with theory, verifying the Zeeman effect

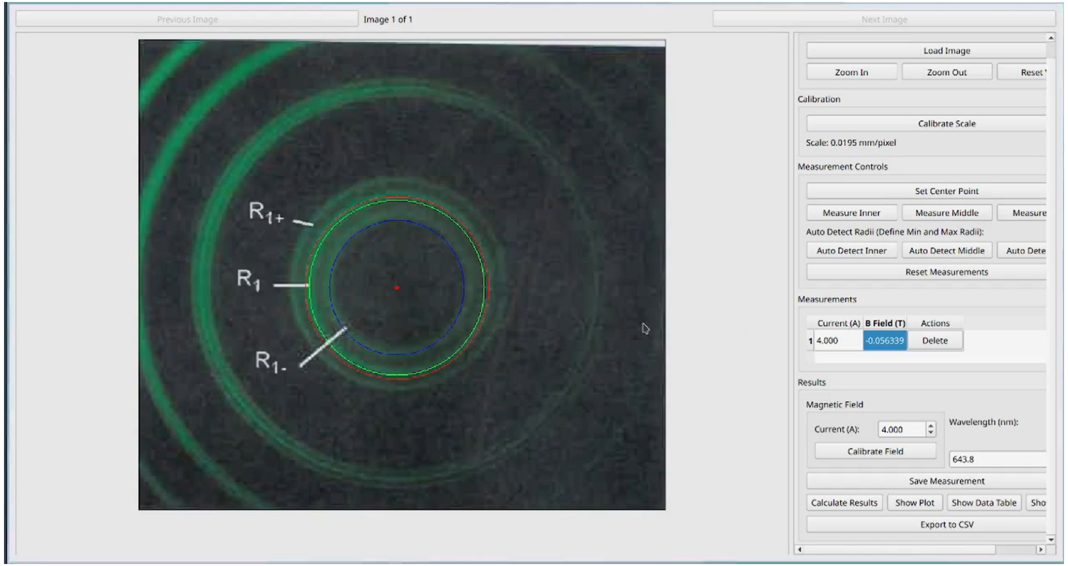


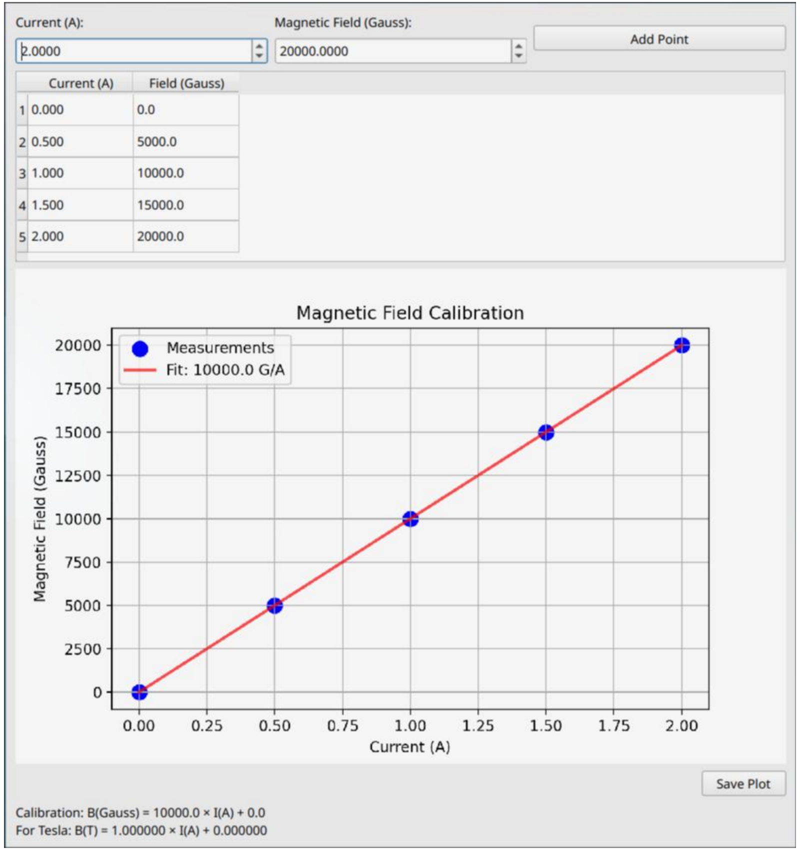
Results



Previous Image
Image 1 of 1
Next Image
R1+
R1
R1-
Image Controls
Load Image
Zoom In
Zoom Out
Reset
Calibration
Calibrate Scale
Scale: Not calibrated
Measurement Controls
Set Center Point
Measure Inner
Measure Middle
Measure
Auto Detect Radii (Define Min and Max Radii):
Auto Detect Inner
Auto Detect Middle
Auto Dete
Reset Measurements
Measurements
Current (A)
B Field (T)
Actions
Results
Magnetic Field
Current (A):
0.000
Calibrate Field
Wavelength (nm):
643.8
Save Measurement
Calculate Results
Show Plot
Show Data Table
Sho
Export to CSV

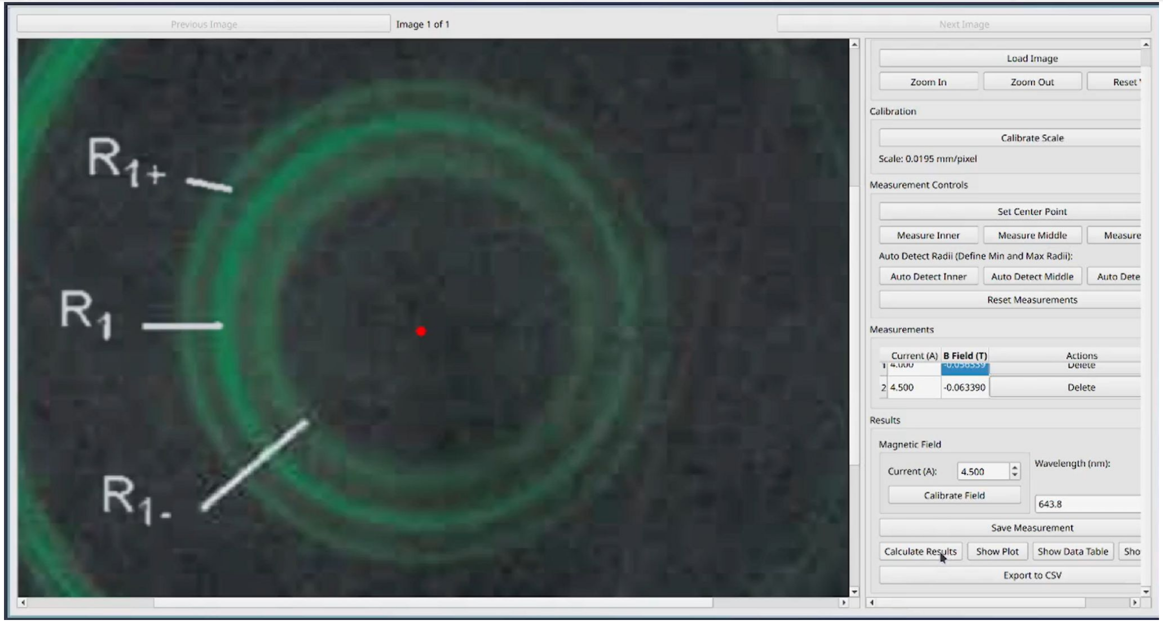
Results (continued)





Results (continued)

	Ri (mm)	Rc (mm)	Ro (mm)	Δλi (nm)	Δλo (nm)	ΔEi (eV)	ΔEo (eV)
1	2.042	2.470	2.937	-0.013	0.017	-3.875e-05	5.065e-05
2	1.887	2.451	3.054	-0.016	0.022	-4.910e-05	6.660e-05



Results (continued)

Final Results:

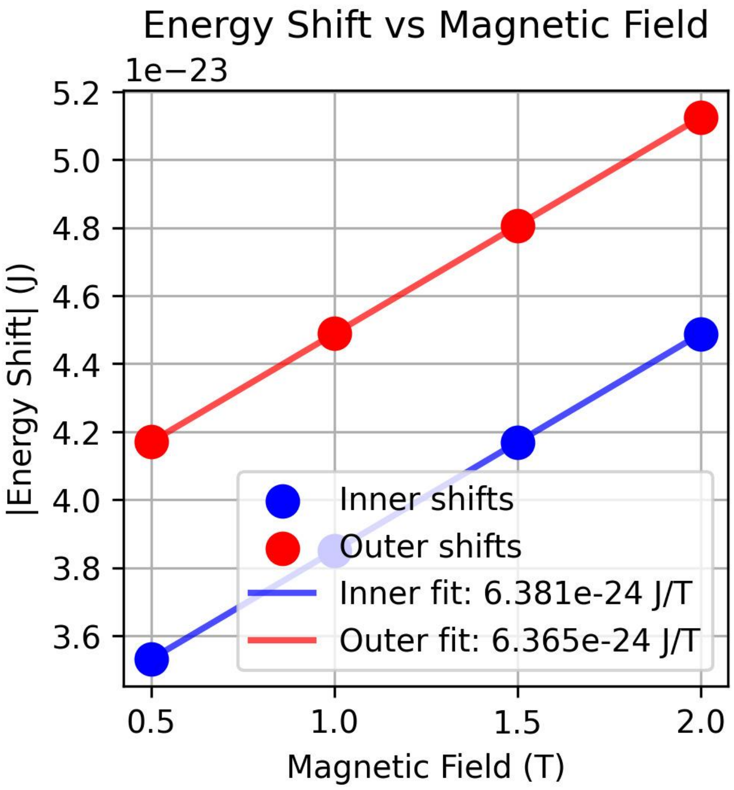
Inner Bohr magneton: 6.381e-24 J/T
Outer Bohr magneton: 6.365e-24 J/T
Average Bohr magneton: 6.373e-24 J/T

Inner specific charge (e/m): 1.210e+11 C/kg
Outer specific charge (e/m): 1.207e+11 C/kg
Average specific charge (e/m): 1.209e+11 C/kg

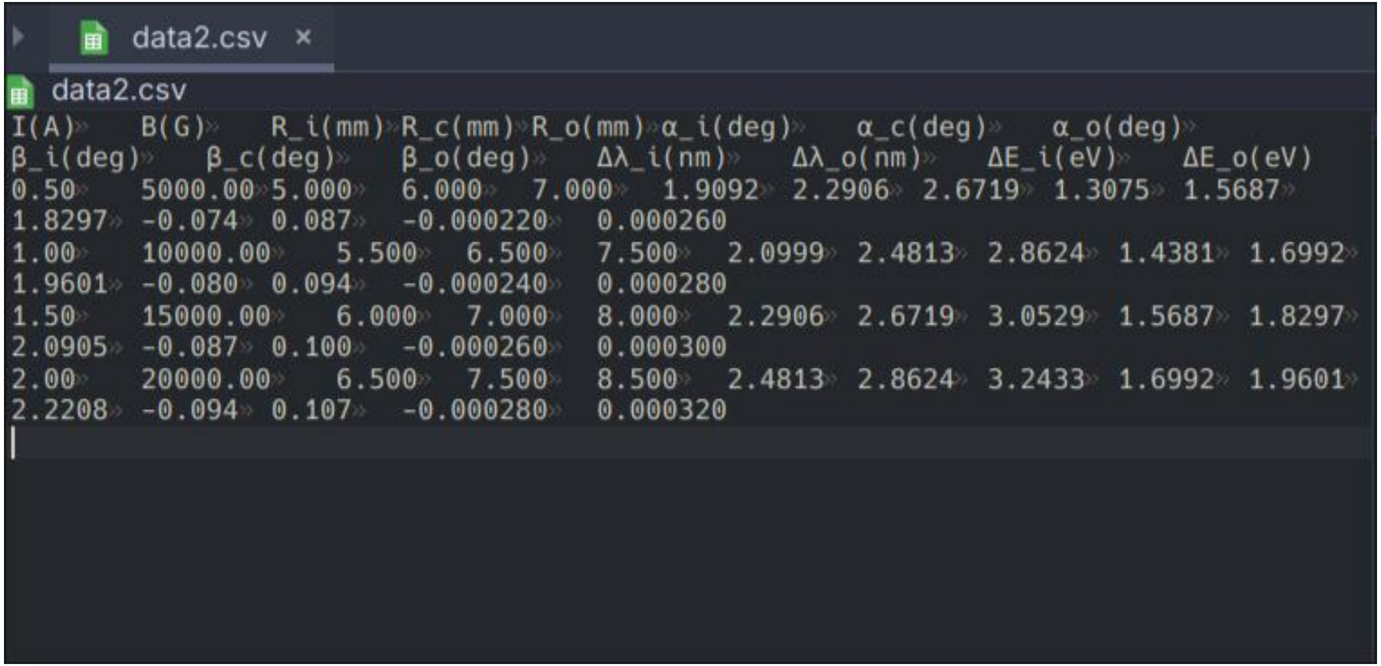
Expected values:
Bohr magneton: 9.274e-24 J/T
Specific charge: 1.758e11 C/kg

Relative errors:
Bohr magneton: 31.3%
Specific charge: 31.2%

Copy Results



Results (continued)



data2.csv ×

data2.csv

I(A)	B(G)	R_i(mm)	R_c(mm)	R_o(mm)	α_i(deg)	α_c(deg)	α_o(deg)	β_i(deg)	β_c(deg)	β_o(deg)	Δλ_i(nm)	Δλ_o(nm)	ΔE_i(eV)	ΔE_o(eV)
0.50	5000.00	5.000	6.000	7.000	1.9092	2.2906	2.6719	1.3075	1.5687	1.8297	-0.074	0.087	-0.000220	0.000260
1.00	10000.00	5.500	6.500	7.500	2.0999	2.4813	2.8624	1.4381	1.6992	1.9601	-0.080	0.094	-0.000240	0.000280
1.50	15000.00	6.000	7.000	8.000	2.2906	2.6719	3.0529	1.5687	1.8297	2.0905	-0.087	0.100	-0.000260	0.000300
2.00	20000.00	6.500	7.500	8.500	2.4813	2.8624	3.2433	1.6992	1.9601	2.2208	-0.094	0.107	-0.000280	0.000320

Thank you