Bloomie: A Smart Robotic Lamp

Nina Binder, Peter Trenkle, Timothy Summers, Leonie Kehlenbeck
LMU Munich
Munich, Germany
l.kehlenbeck@campus.lmu.de,nina.binder@campus.lmu.de,t.summers@campus.lmu.de,p.trenkle@campus.lmu.de

Abstract

Traditional lamps are inflexible, requiring manual adjustment for different lighting conditions. To overcome this, we developed **Bloomie**, a smart robotic lamp that supports gesture control, follow mode, and pre-set postures for adaptive and user-centric lighting. Developed based on the MyCobot280 robotic arm, Bloomie features a 3D-printed lamp body with embedded LEDs and utilizes Mediapipe for hand recognition, ROS for communication, and Python for robot control.

Our system enables users to switch between interaction modes, dynamically follow hand movements, and regulate brightness via natural gestures. Testing confirmed that gesture recognition is reliable, though angle calculations occasionally introduce imprecision. Despite this, Bloomie demonstrates how robotics can be used to augment everyday items, making lighting more adaptive and interactive. This work showcases the potential of human-robot interaction (HRI) in smart home settings, paving the way for more intuitive and responsive assistive technologies.

CCS Concepts

• **Human-centered computing → Human computer interaction (HCI)**.

Keywords

human computer interaction, intelligent lamp, gesture-based control, adaptive lighting, robotic arm, myCobot280, human-robot interaction, gesture recognition, mediapipe, ROS

1 Introduction

In recent years, the integration of robotics into everyday environments has gained significant attention. Robotic systems are increasingly being designed to assist users in daily tasks, offering improved automation, adaptability, and intuitive interaction. Among these applications, robotic lighting systems provide a novel approach to personalized illumination, dynamically adjusting to user preferences and environmental conditions.

Traditional lamps offer limited interactivity, often requiring manual adjustments for positioning and brightness control. In contrast, an intelligent robotic lamp can provide hands-free control, adapting its position and light settings based on user input. Using advanced computer vision and robotic control, such a system enhances usability, accessibility, and efficiency.

This paper presents the development of a robotic smart lamp that integrates computer vision, gesture recognition, and robotic arm control. The system is designed to respond to user gestures, adjusting its position and light settings accordingly. To achieve this, we employ the MyCobot 280 robotic arm in combination with the Robot Operating System (ROS), Mediapipe for gesture recognition, and an ESP8266 microcontroller to manage lighting functions. The system operates in four distinct modes: gesture mode, follow mode, light mode, and posture mode.

A major challenge in this project was to achieve reliable gesture recognition and translate these inputs into precise robotic movements. Ensuring smooth and responsive control required careful optimization of the vision processing pipeline and motion commands. Furthermore, MyCobot 280's GPIO pins were inaccessible, making it necessary to integrate an external microcontroller for lighting control.

The remainder of this paper provides an in-depth discussion of the system architecture, implementation details, and the technical challenges encountered during development. Our approach demonstrates the feasibility of combining robotics and human-computer interaction to create an adaptable and user-friendly lighting solution.

2 System Overview

2.1 Hardware

The system integrates multiple hardware components to enable gesture-controlled lighting using a robotic arm. Each element serves a distinct function in achieving seamless user interaction and dynamic lighting control.

2.1.1 MyCobot 280 M5. The MyCobot 280 M5 robotic arm serves as the primary hardware platform in this project. It is connected to a computer via USB and controlled through ROS (Robot Operating System) nodes that execute movement commands based on hand gestures. The robotic arm remains unmodified, using its standard configurations and built-in LEGO-compatible mounting points for easy attachment of additional components.

2.1.2 Lamp Head. The lamp head features an adjustable lens system that allows users to switch between ambient and focused lighting. The lens moves up or down to modify the light cone, creating either a wide, soft illumination for relaxation or a narrow, concentrated beam for task lighting.

To achieve this movement, the design employs a helical groove mechanism similar to that used in zoom lenses. The system consists of:

- An outer tube with a helical cam track.
- An inner tube with an opposing track.
- A guiding pin fixed to the lens holder that engages both tracks.

When the outer and inner tubes rotate relative to each other, the opposing grooves force the guiding pin, and thus the lens, to

move up or down with high precision. This setup enables smooth transitions between lighting modes.

The lens mechanism is adjusted using the top rotating joint of the robotic arm, eliminating the need for additional motors. The lamp head is 3D printed, allowing for rapid prototyping and complex geometry while maintaining a lightweight structure. It attaches to the robot arm using LEGO compatible pins, ensuring plug-and-play functionality.

2.1.3 Micro D1 Mini + 3 LED Rings. To enable lighting control, a D1 Mini microcontroller with an ESP chip processes lighting commands received from the computer. The LED rings are pre-programmed to interpret incoming data, adjusting brightness and color accordingly.

The microcontroller serves two functions:

(1) Signal interpretation – It decodes incoming commands sent from the ROS nodes via serial communication.
(2) Power distribution – It supplies power to the LED rings, eliminating the need for an external power source.

A single additional data cable is used to address individual LEDs within the three interconnected LED rings, allowing for potential future expansion in dynamic lighting effects.

2.1.4 Camera. The camera is responsible for gesture detection, capturing the user's hand movements to allow real-time interaction with the robotic arm. Any standard webcam can be used, as no specific model is required. For this project, the camera is positioned in front of the user to ensure clear visibility of hand gestures. An alternative setup, where the camera is mounted on the robotic arm, was considered but was ultimately not implemented due to additional complexity.

2.2 Software

Our project integrated various software tools to enable gesture-based robotic control and hardware interaction, each playing a crucial role in system functionality.

We used Google's Mediapipe for real-time hand tracking, leveraging 21 hand landmarks to extract position data and interpret movement directions for robotic control.

The Robot Operating System (ROS) acted as the project's communication system, between the gesture detection and robot control modules to ensure smooth data transfer.

The Pymycobot API in Python was used to directly control the MyCobot 280 robotic arm, allowing detected gestures to be translated into precise joint movements.

With OpenCV (CV2), we visualized the camera feed with hand-tracking overlays, providing real-time feedback to assist in debugging and refining gesture-based commands.

Since the MyCobot 280's GPIO pins were inaccessible, we used an external microcontroller, programmed via Arduino IDE, to control LED rings, adjusting lighting patterns based on gesture inputs.

3 Key Features

4 System Architecture

Our system is built around three distinct ROS (Robot Operating System) nodes: the *Camera Node*, the *Robot Node*, and the *Light Node*. Each of these nodes has a specific role in the system, enabling modular and efficient control of the robotic lamp.

4.1 Camera Node

The Camera Node is responsible for handling all camera-related functionality. It includes three key detection components:

- **Mode Detector:** Detects activations and deactivations of various modes based on visual input.
- **Gesture Detector:** Recognizes hand gestures for Gesture Mode.
- **Follow Detector:** Tracks motion and identifies targets for Follow Mode.

Each detector operates independently and publishes data to separate communication channels, allowing other nodes to process relevant information in real-time.

4.2 Robot Node

The Robot Node serves as the main controller for the robotic arm. It subscribes to the channels published by the Camera Node and processes incoming data to determine the necessary movements.

To handle different functionalities, the Robot Node utilizes separate control files, each dedicated to a specific mode:

- `posture_control`: Handles predefined postures and positioning of the robot.
- `follow_control`: Implements movement logic for tracking a target.
- `gesture_control`: Interprets hand gestures and maps them to robotic actions.

Each control file implements the necessary robotic movement using the MyCobot API functions to ensure precise and responsive behavior.

4.3 Light Node

In Light Mode, the system also needs to control the lighting mechanism. The Light Node is responsible for managing the light logic via an Arduino board. It receives activation signals from the Robot Node and triggers the appropriate commands to switch the light on or off as required.

By structuring the system into these three ROS nodes, we achieve a clear separation of concerns, making the architecture more modular, maintainable, and scalable.

5 Implementation

5.1 Modes

The smart lamp system has four different modes, each intended to improve the functionality and flexibility of the robotic arm. They provide intuitive interaction and dynamic control of the lamp. The modes include:

- **Mode Gesture:** Allows users to control the lamp's movement using directional gestures such as *up*, *down*, *left*, and *right*. The mode offers manual control over the position of the lamp. It becomes active when the index, middle, and ring fingers are extended, but the thumb and pinky are folded.

- **Mode Follow:** Enables the lamp to dynamically track hand movements in real-time. In this mode, the lamp follows the user's hand, illuminating objects of interest automatically. This mode is activated when the thumb and pinky fingers are extended.
- **Mode Light:** Allows users to turn the light on or off and adjust its brightness or color settings through a specific gesture. It is activated when the index and pinky fingers are extended.
- **Mode Posture:** To toggle between pre-set positions for ambient or directed lighting. It is activated when all fingers except the thumb are extended.

Mode detection is performed using Mediapipe hand tracking with the `ModeDetector` class. Modes need to be deactivated before enabling one to avoid conflicting states.

5.1.1 Mode Gesture. Mode Gesture provides the user with accurate control of lamp motion through interpreting direction hand gestures. This mode benefits from Mediapipe's real-time 21 landmarks tracking on the hand, focusing on the wrist (landmark 0) and the index finger (landmarks 5 and 8).

For detecting direction gestures, the system calculates angles in 3D space from vectors formed between these prominent points: wrist → index base → index tip. The calculated angles are used to determine specific movements:

- A positive slope along the y-axis indicates an upward gesture (*up*).
- A negative slope along the y-axis indicates a downward gesture (*down*).
- A positive slope along the x-axis indicates a rightward gesture (*right*).
- A negative slope along the x-axis indicates a leftward gesture (*left*).

To ensure reliable detection, gestures must be consistent across multiple frames before being confirmed. This stabilizing process prevents false positives because of transient hand movements or noise from the camera.

5.1.2 Mode Follow. The follow logic tracks hand movements and translates them into real-time motion commands for the MyCobot 280 robotic arm. Using Mediapipe's hand-tracking model, the system detects the hand's position, compares it to the last known location, and calculates a two-dimensional movement vector. This vector directs the robot to adjust its joint angles, allowing it to "follow" the user's hand.

To ensure stability, a tolerance box is centered on the screen. Minor hand movements inside this box are ignored, preventing unnecessary micro-adjustments. If the hand moves outside, the system calculates a displacement vector and sends it to the robotic arm.

Predefined joint constraints ensure safe operation, preventing excessive movement. The ground joint's horizontal range is limited to -50° to 90°, though these values may require adjustments based on calibration differences among MyCobot 280 units.

5.1.3 Mode Light. Mode Light introduces a gesture-based approach to controlling the robotic lamp's lighting properties, including power state, brightness, and color selection. Since the MyCobot 280's GPIO pins were inaccessible, an ESP8266 microcontroller was integrated to handle the LED system via USB serial communication. This allows for independent lighting control without interfering with the robot's movement functions.

The hardware consists of three interconnected LED rings, which were soldered together and powered directly by the ESP8266. This eliminates the need for an external power source while ensuring stable operation. The microcontroller listens for serial commands from ROS and updates the LED state accordingly.

To manage communication between ROS and the ESP8266, two dedicated ROS nodes were developed:

- `light_control.py` – Processes gesture-based lighting commands, allowing users to turn the light on and off. The logic for color selection is implemented but not yet fully controllable.
- `light_connector.py` – Translates ROS messages into serial commands and sends them to the ESP8266, which modifies the LED output.

While Mode Light enables users to toggle the light via a specific hand gesture, color switching is not yet functional. The necessary logic is in place, meaning future updates could unlock full-color control. The system currently supports real-time brightness adjustment, but for now, the lamp remains limited to simple on/off functionality.

By separating lighting control from robotic movement, Mode Light ensures stable operation and intuitive user interaction. Future improvements may include WiFi-based control, dynamic lighting effects, or adaptive lighting based on environmental conditions.

5.1.4 Mode Posture. When the posture mode is activated, the lamp moves to a predefined position. Once this position is reached, posture mode is deactivated, and the lamp is open to new commands.

This allows users to quickly switch between different lamp postures without manually adjusting them. There are two postures:

- **Ambient:** All angles of the robotic arm are set to 0, except for the last joint, which is angled 90° upwards. This results in an upward-facing desk lamp that provides ambient lighting.
- **Directional:** The arm angles are slightly offset, positioning the lamp at an angle to the side. This allows for more focused lighting on a specific object and gives the lamp a more traditional desk lamp appearance.

The `send_angles` function from the MyCobot API is used to move the lamp to the predefined position.

5.2 Robot Movement

The robot is connected via USB and controlled using the MyCobot API. To execute different types of movements, we utilize the following functions:

- `send_angles(degrees, speed)`
  - This function moves the robotic arm to a specified set of joint angles at a given speed, allowing for precise positioning.
  - **Used for:** Posture control
- `jog_angle(joint_id, direction, speed)`

  - This function enables continuous movement of an individual joint by specifying its ID, direction, and speed, making it useful for real-time adjustments.
  - **Used for:** All continuous movement
- `jog_stop()`
  - This function stops any ongoing movement initiated by `jog_angle`, ensuring controlled and immediate halting of motion.
  - **Used for:** Stopping continuous movement

6 Challenges and Lessons Learned

6.1 Challenges

One of the first challenges was finding a stable virtual machine that could run the ROS ecosystem on an ARM-based MacBook. This setup was necessary to interface with the MyCobot 280 robotic arm and a webcam via USB ports. After testing different solutions, we found that the UTM application provided the best stability and compatibility for our requirements.

After successfully connecting the robotic arm to the virtual machine, another challenge arose: controlling the joint angles of the robot using gestures. While gesture recognition worked reliably, translating hand movement data into precise robot joint angles proved to be error-prone. The transformation process introduced small inaccuracies, often leading to unintended or imprecise movements.

Our initial approach to robot movement relied on MoveIt, a widely used motion planning framework for ROS. However, we quickly discovered that MoveIt was not well-suited for real-time gesture-based control. Its motion planning algorithms, designed for predefined paths and collision avoidance, resulted in noticeable delays and sluggish responses, making it impractical for our interactive control needs.

A major limitation we encountered was the inability to interface with the MyCobot 280's GPIO pins. Our project involved creating a lamp system that incorporated LED rings and a microcontroller, allowing us to control the lighting using hand gestures. Since the MyCobot 280 has GPIO pins, we initially planned to connect the LED rings directly to the robot, eliminating the need for an external microcontroller. However, due to a complete lack of documentation and error feedback, we were unable to establish direct communication with the pins. This forced us to rely on an external microcontroller instead.

6.2 Lessons Learned

Virtualization can be challenging for robotics but is feasible. Running ROS on an ARM-based MacBook was initially difficult, but UTM provided a stable and workable solution.

Keyboard control was essential for debugging. Before fully implementing gesture control, using keyboard inputs for movement helped identify and fix issues more efficiently.

Using the MyCobot API for direct joint control is better than motion planning for real-time interaction. MoveIt was too slow for real-time gesture-based control, but the API provided immediate responsiveness, making it a much better choice.

Our attempt to connect LED rings directly to the MyCobot 280's GPIO pins failed due to a lack of documentation and error feedback. In future projects, ensuring comprehensive documentation and alternative communication methods would be crucial for hardware integration.

7 Future Work

While the current implementation is robust in functionality, there are several areas where accuracy and extension can be achieved to improve the intelligent lamp further. One such area is improving gesture recognition accuracy for directional gestures such as *up*, *down*, *left*, and *right*. This would improve interactions to be smoother and more uniform, especially in dynamic scenarios.

Another potential extension is the addition of voice commands as an alternative input method. This would provide access and make users feel more in control of the lamp.

Adding the camera component to the robotic arm could improve tracking quality by allowing the system to respond to objects in its surroundings dynamically without requiring an external static camera.

Direct integration of the light circuit with the robotic arm is another possible update. This upgrade would simplify the system design and also make it more portable.

Additional modes could also be implemented to enhance functionality. An example of this is a "Party Mode," which could have dynamic lighting effects, and a "Pomodoro Timer Mode," which would use timed changes in lighting to facilitate focused work or study sessions.

These proposed improvements are aimed at making the smart lamp more interactive, versatile, and user-friendly in finding new uses beyond its current functionality.

8 Conclusion

In this project, we successfully developed **Bloomie**, an intelligent robotic lamp that adapts to user needs through gesture-based interaction and dynamic lighting control. By leveraging the MyCobot280 robotic arm and technologies such as Mediapipe, ROS, and Python, we created a system capable of providing both ambient and focused lighting while offering intuitive control through gestures. The implementation of four distinct modes—Gesture, Follow, Light, and Posture—demonstrates the versatility of the system in addressing various use cases.

Our work highlights the potential of integrating robotics into everyday objects to enhance their functionality and interactivity. While the system performs well in its current state, there is room for improvement. Overall, Bloomie represents a step toward making lighting solutions more adaptive and user-friendly.

References

A Character Count

This work contains 18937 characters.