College of Engineering
Mechanical and Mechatronics Engineering Department
MECA 440- Microcontrollers for Mechatronics
Fall 2023

Final Project

PID Light Tracker using Arduino

Rami Kronbi - 20210359
Wassim Ghaddar - 20210586

Dr. Ahmad Koubeisi

[December 13, 2023]

Table of Contents

Project Description 3

Functional Specifications 3

Block Diagram 5

Principle of Operation 6

  Light Sensing 6

  Crossing Operation 6

  Error Calculation 7

  PID Control Algorithm 7

  Servo Motor Control 7

  Principle of Operation Summary 8

Schematic Diagram 8

Bill of Quantity 8

Results and Conclusion 9

Appendix 12

Project Description

The goal of this project is to build a robot capable of autonomously tracking a light source using PID (Proportional-Integral-Derivative) control. The robot will be equipped with two servo motors that control its yaw and pitch movements, allowing it to adjust its orientation and track the light source accurately. The system will include components such as a robot chassis, servo motors, a light sensor, a microcontroller, and a power supply. The microcontroller will process the sensor readings, calculate the appropriate servo movements using the PID control algorithm, and generate the control signals for the servo motors. By implementing this project, you will create a robot that can track a light source with precision and learn about concepts such as PID control, sensor integration, and servo motor control.

This light-tracking robot project offers a practical application of PID control principles and robotics. The robot's ability to track a light source using servo motors and a light sensor showcases the integration of hardware and software components. Through the development process, you will gain insights into assembling the robot chassis, attaching servo motors, connecting the light sensor, and programming the microcontroller. Additionally, you will have the opportunity to calibrate and fine-tune the PID parameters to optimize the tracking performance. This project provides a hands-on learning experience in control systems, sensor integration, and robotics, enabling you to deepen your understanding of these concepts and explore further applications in the field of robotics and automation.

Functional Specifications

1. Chassis Design:
    - The robot chassis was designed to accommodate both servo motors, light sensor modules, and the microcontroller, along with the wire managing techniques.
    - The material chosen for the design was 3d printed PLA(Polylactic acid) as it offers a lightweight yet sturdy build to allow for easy movement of the Servo Motors without putting too much load on the gears of the motors along with stable directing of the tilt tracking mechanism.
    - The light sensor modules had to be physically on the surface of the mechanism to fine tune their sensitivity via the potentiometer knob on the module as well as not hindering the proper receiving of the signal from the light source.

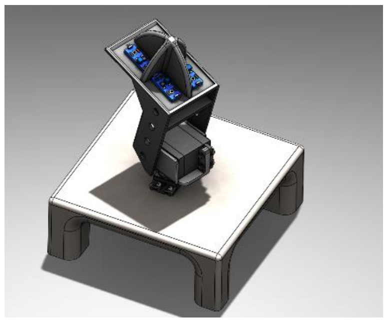


Figure 1: 3d Model of The Design

2. Servo Motors:
    - Two servo motors will be used for controlling the yaw and pitch movements of the robot.

    - The servo motors had to ensure sufficient range of motion to cover a wide field of view for effective light source tracking. The motors must be responsive and precise in their movements.

    - That’s why the choosing of 2 Mg995 Servo Motor which deliver higher torque values along with precise angle movements capable of the range of motion.

3. LDR Sensor Modules
    - The light sensor will be responsible for detecting the intensity of the light source.

    - The capability of providing accurate readings that correspond to the light intensity detected by the sensor was crucial. Thus, the choosing of LDR modules was a better option than normal LDR sensors as it offers the variability in the sensor’s sensitivity and allows for optimum calibration under any conditions.

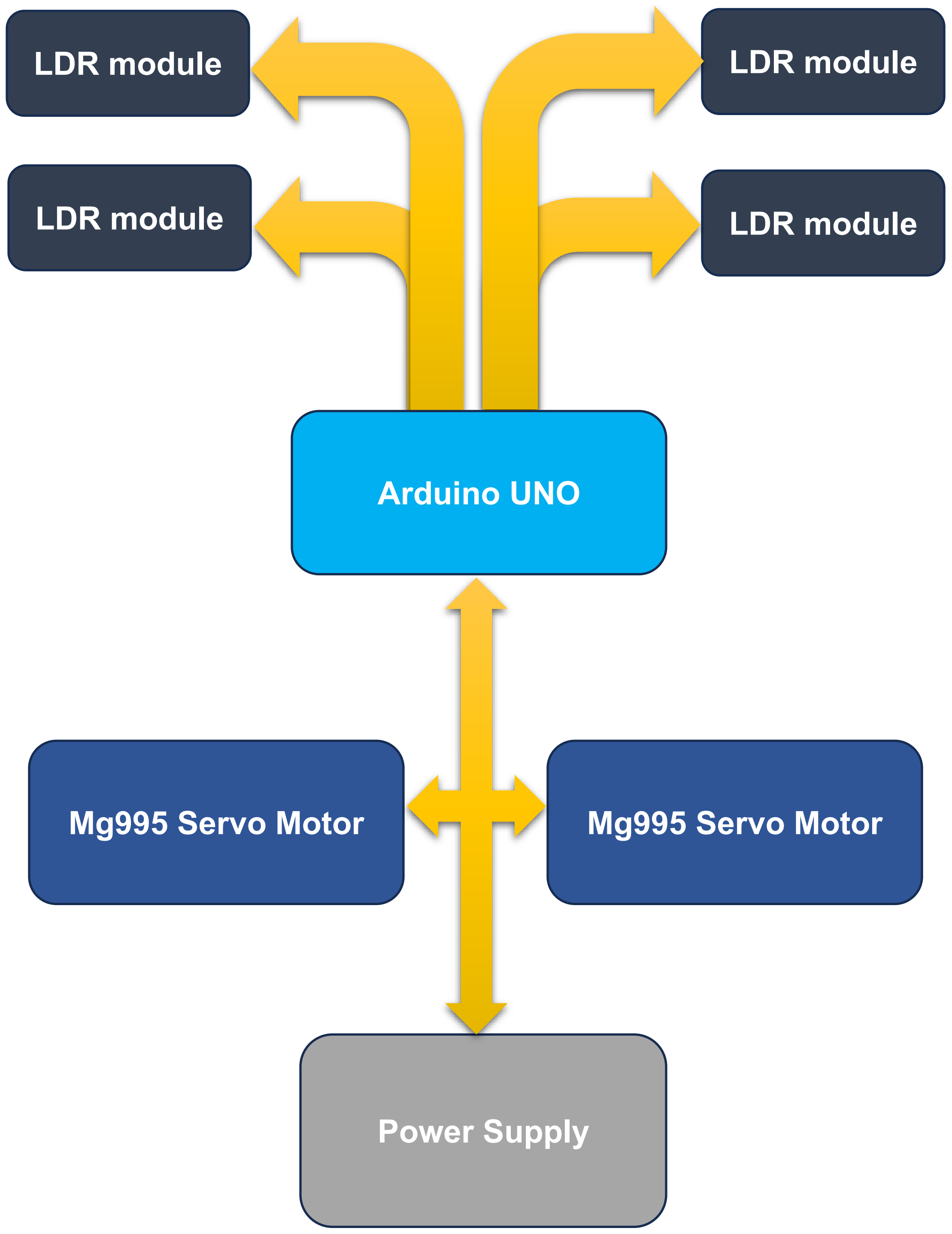
4. Microcontroller
    - The choice of microcontroller was dependent on easiness of control, programming, and applying fundamental PID control algorithms, therefore the choice of the Arduino Uno was a better choice than for example and Arduino nano which sometimes causes unexpected problems.

5. PID Control Algorithm
    - The PID control algorithm was implemented to calculate the optimum angle movements for the servo motors based on the difference of the averages of the top and bottom sensors along with the right and left sensors. This ensured proper functionality and produced the most accurate readings from the sensors. Especially, that this system works on 2 Degrees of Freedom and any slight deviation or inaccuracies might cause the system to respond differently than intended to.
    - Fine-tuning parameters of PID(proportional , integral, derivative) gains was done to achieve smoothness and fluidity of the system’s functionality.

6. Power Supply
    - An external power supply was used to provide stable and sufficient voltage for the Servo Motors, LDR modules, and the Arduino UNO microcontroller.

    - The choice of an external power supply was important to ensure stable voltage was being delivered to the LDR’s and to eliminate any chance of disturbances due to technical restrictions.

Block Diagram



Principle of Operation

The core principle lies in harnessing the power of PID (Proportional-Integral-Derivative) control with a dual-axis system. Equipped with servo motors governing yaw and pitch movements, this autonomous system is designed to dynamically adjust its orientation, ensuring seamless tracking of a light source.

This project introduces many techniques in controlling, adjusting, and integrating the system to reduce the percentage of error as much as possible. The principle of operation begins with:

Light Sensing

- The light sensor plays the pivotal role as the sensory apparatus of the system. Functioning as an optical transducer, it converts incident light intensity into an electric signal.

- This signal is a numeric reflection of the amount of light intensity of the surrounding

- This signal is an analog value which is allows for easier differentiation in the amount of light intensity being provided across each of the four LDR sensors as they are provided in bits (0 to 1023) which governed the resolution of the Analog-to-Digital converter in the Arduino Microcontroller.

Crossing Operation

Before the Microcontroller engages in the task of error calculation, it first executes an operation known as crossing.

- Crossing is an operation that involves retrieving data from the four sensors that are positioned precisely on the top plate and calculating the average light intensity that crosses spatially between these exact distributed points

- Keeping in mind this system achieves a level of stability in ambient light conditions just as any other luminous conditions and this is due to the crossing technique which calculates difference in average between all four of the sensors

- Averaging helps the system to be adaptable to any light condition and equips the system with the ability to understand the ambient light distribution,

- By providing a more representative value of the overall luminosity surrounding of the robot, accurate, precise, and promising results could be achieved.

Error Calculation

The essence of precision in the light-tracking robot project lies in the microcontroller's adept handling of error computation. The microcontroller calculates the error based on the results obtained from the crossing technique mentioned above.

- The microcontroller calculates the **ERROR** by subtracting the desired light intensity which is the **SETPOINT** from the actual intensity being calculated from the average being calculated during the crossing method .
- The error represents the deviation of the system from the optimal condition
- The optimal condition in our case is getting as much light intensity being directed towards the sensor since the difference in averaging would be minimal and error would tend to zero
- Corrective actions are taken later PID control Algorithm to achieve this optimal condition

PID Control Algorithm

- **Proportional (P):** The P component or (**Kp**) scales the error directly to determine the required correction. A higher error results in a larger corrective response. This component is crucial for initial and rapid adjustments, and it is the usually referred to as the ***GAIN***
- **Integral (I):** The I component or (**Ki**) accumulates the error over time, addressing any persistent deviation from the setpoint. It helps eliminate steady-state error caused by the gain being delivered and ensures long-term stability in the system.
    - Note: This component was adjusted in a way to not accumulate any error for more than 10 seconds due to saturation reasons.
- **Derivative (D):** The D component or (**Kd**) anticipates the future trend of the error by considering its rate of change. This helps prevent overshooting and oscillations, enhancing the system's response.

Servo Motor Control

- After error calculation and applying principal PID control algorithm, the microcontroller translates the output from the PID algorithm into control signals for the servo motors.
- These Signals are known as PWM (Pulse-Width-Modulation)
- These signals determine the angle or position adjustments required for both yaw and pitch movements based on retrieved values from the PID algorithm.

Principle of Operation Summary

By maintaining this closed-loop feedback system, the robot adapts to changing conditions, ensuring robust and accurate light tracking. The PID algorithm's dynamic nature, incorporating both immediate and accumulated corrective measures, enhances the overall performance of the system, making it responsive and adaptable in real-world scenarios.

Schematic Diagram

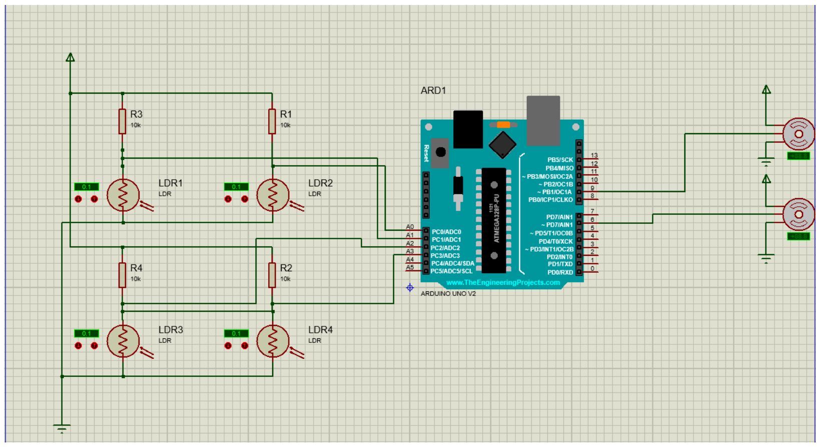
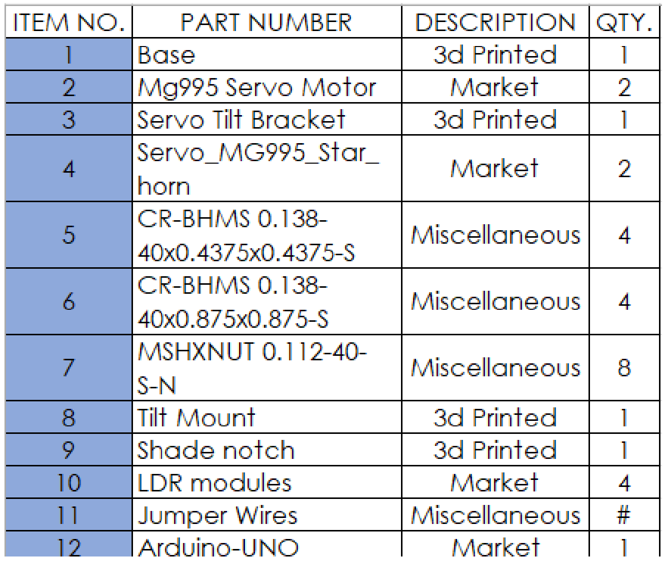


Figure 2: Schematic Diagram showing the Circuitry on Proteus

Bill of Quantity



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Base	3d Printed	1
2	Mg995 Servo Motor	Market	2
3	Servo Tilt Bracket	3d Printed	1
4	Servo_MG995_Star_horn	Market	2
5	CR-BHMS 0.138-40x0.4375x0.4375-S	Miscellaneous	4
6	CR-BHMS 0.138-40x0.875x0.875-S	Miscellaneous	4
7	MSHXNUT 0.112-40-S-N	Miscellaneous	8
8	Tilt Mount	3d Printed	1
9	Shade notch	3d Printed	1
10	LDR modules	Market	4
11	Jumper Wires	Miscellaneous	#
12	Arduino-UNO	Market	1

Figure 3: 3d Model of The Design

Results and Conclusion

Upon completion of the light-tracking robot project, several key outcomes were observed, showcasing and effectiveness of the implemented design and control system.

❖ **Results**

1. **Tracking Precision:** The robot demonstrated remarkable precision in tracking the light source accurately and adjusting its orientation through the coordinated movement of the servo motors upon any input and change from the light source.

2. **Adaptability to Light Conditions:** The light sensor proved to be a reliable component, showcasing consistent performance across a spectrum of lighting conditions. This adaptability was crucial in ensuring the robot's ability to effectively track the light source under diverse environmental settings.

   a. An Example on that includes the use of a lamp as a light source in a dark room and using the sunlight in a well-lit environment to ensure seamless tracking in both conditions

3. **Structural Stability: The robot demonstrates** an overall structural integrity with the use of a robust chassis that is designed to be as lightweight and tough as possible. With the Use of CURA software to adjust the infill concentration of the base and the 3d printed parts, we ensured the stability of the whole system.

4. **Optimized PID Parameters:** Through a systematic process of calibration and fine-tuning, the PID parameters were optimized. Achieving a delicate balance between proportional, integral, and derivative gains was pivotal in enhancing the overall tracking performance, reducing overshooting, and minimizing response delays.

❖ **Discussions/ Problems Faced**

During the development of the light-tracking robot, one significant hurdle we encountered revolved around the use of LDR (Light Dependent Resistor) sensors on the PCB. The initial design incorporated individual LDR sensors, aiming for a cost-effective solution. However, this approach posed challenges due to the inherent limitations of standalone LDR sensors, primarily related to accuracy and sensitivity.

1. **Inaccuracy in Light Detection:** Standalone LDR sensors exhibited inconsistencies in detecting light intensity accurately. The variations in ambient light conditions led to challenges in achieving the desired precision in light tracking.

2. **Limited Sensitivity:** The sensitivity of individual LDR sensors was not optimal for the dynamic and real-time adjustments required for effective light tracking. This limitation

became evident during testing, where the robot struggled to respond promptly to rapid changes in light intensity.

3. **Complex Calibration Requirements:** Calibrating individual LDR sensors to achieve uniformity in response across different units proved to be a complex and time-consuming task. The lack of consistency hindered the overall performance of the robot.

In response to these challenges, a pivotal decision was made to transition from individual LDR sensors to LDR modules. This shift to LDR modules proved to be a game-changer in improving the accuracy and reliability of the light-tracking system.

This experience highlighted the importance of careful consideration when selecting and integrating sensors into the PCB design. While standalone LDR sensors may be suitable for certain applications, the specific requirements of the light-tracking robot needed a more sophisticated solution. The transition to LDR modules not only addressed the challenges faced but also contributed to an overall improvement in the performance and reliability of the robot's light-tracking capabilities.

1. **Enhanced Accuracy and Consistency:** LDR modules provide a more reliable and consistent measurement of light intensity. The modules, with integrated components for signal conditioning, ensured a standardized and accurate response across different units of the robot.

2. **Improved Sensitivity:** The use of LDR modules with enhanced sensitivity allowed the robot to detect subtle changes in light intensity more effectively. This improvement was crucial for the robot's ability to track the light source smoothly and precisely.

3. **Simplified Calibration Process:** The integration of LDR modules streamlined the calibration process. With inherent consistency in the response characteristics of the modules, the calibration became more straightforward, reducing the time and effort required to fine-tune the system.

❖ Conclusion

In wrapping things up, building the light-tracking robot taught us a lot. We started with some issues using basic light sensors (LDR), but we learned they weren't accurate enough. So, we switched to better sensors called LDR modules, which made a big difference.

Through the ups and downs, we realized that picking the right sensors is crucial. The changes we made improved how the robot follows light, making it more accurate and responsive.

This project wasn't just about building a robot; it was a hands-on lesson in how things work in the real world. Now, our robot smoothly follows the light, and it's ready for more exciting possibilities in the world of robots and automation. It was a journey with challenges, but in the end, we've got a cool, reliable light-tracking robot.

Appendix

1. Code
    a. INO FILE

```
//////////////////////////////Libraries Used////////////////////////////////
  #include "LightPID.h"
  #include <Servo.h>
////////////////////////////////////////////////////////////////////////////

//////////////////////////////Servo Variables///////////////////////////////
  Servo panServo;
  Servo pitchServo;
  const int panPin = 9;
  const int pitchPin = 6;
////////////////////////////////////////////////////////////////////////////

//////////////////////////////Sensor Values/////////////////////////////////
  double botr, botl, topr, topl;
  double setpoint, feedback;
  double output;
  double curError;
  int initAngle =0;
  int tiltAngle =0;
  bool orientation;// 0 for [0, 180] / 1 for [180, 360]
////////////////////////////////////////////////////////////////////////////
float originTime, curTime;
//////////////////////////////PID Objects///////////////////////////////////
  Pan pan(0.075, 0.00045, 0.00035, -90, 90); //0.1, 0.00018, 0.0007
  Pitch pitch(0.03 , 0.00018, 0.00035, -70, 55); //0.07, 0.00018, 0.002
////////////////////////////////////////////////////////////////////////////

void setup() {
  Serial.begin(9600);

  pinMode(panPin, OUTPUT);
  pinMode(pitchPin, OUTPUT);

  setpoint = 0;
  feedback = 0;
  curError = 0;

  panServo.attach(panPin);
  pitchServo.attach(pitchPin);

  //Initial servo positions
  panServo.write(90);
  pitchServo.write(90);
  //delay(9000000);
originTime = millis();
}

void loop() {
  //if ((millis()-originTime) >10000) pan.integReset();
  //////////////////////////////Control of the tilt
mechanism/////////////////////////////////
  //pitch control
  readSensors();

  //Calculating the systems input and feedback
  setpoint = (botr + botl)/2.00;
  feedback = (topr + topl)/2.00;

  //Calculating error
  curError = setpoint - feedback;
  pitch.setError(curError);

  //Using the calculate method to get the output of the PID
  output = pitch.calculatePID();

  //Determening the hemisphere in which the tilted system is currently located
  if (output > 0) orientation = true;
  else if (output < 0) orientation = false;

  pitchServo.write(90 - output);

  //////////////////////////////Control of the pan
mechanism/////////////////////////////////
  readSensors();
  setpoint = (botr + topr)/2.00;
  feedback = (botl + topl)/2.00;

  curError = setpoint - feedback;
  pan.setError(curError);

  output = pan.calculatePID();

  //If the system is the the [180, 360] hemisphere, the output is inverted
  if (orientation == true) output = -output;
  panServo.write(90 - output);
}

void readSensors(){
  topr = analogRead(A0);
  topl = analogRead(A5); //change to A1
  botr = analogRead(A2);
  botl = analogRead(A3);
}
```

b. Header File

```
class Pan {
  public:
  double kp, kd, ki;
  double integError, derivError, error, prevError, min, max;

  // constructor
  Pan(double kp, double kd, double ki, double min, double max){
    this->kp = kp;
    this->kd = kd;
    this->ki = ki;
    integError = 0;
    derivError = 0;
    error = 0;
    prevError = 0;
    this->min = min;
    this->max = max;
  }

  void setError(double error){
    this->error = error;
  }

  void integReset(){
    this->integError = 0;
  }
  double calculatePID(){
    double output;
    derivError = this->error - prevError;
    integError += this->error;
    output = kp*this->error + kd*derivError + ki*integError;
    prevError = this->error;
    if (output > max) output = max;
    else if (output < min) output = min;
    return output;
  }
};

class Pitch : public Pan{
  public:
  Pitch(double kp, double kd, double ki, double min, double max) : Pan(kp, kd, ki, 
min, max) {}
};
```