Guide for Building a Mobile Robot Manipulator

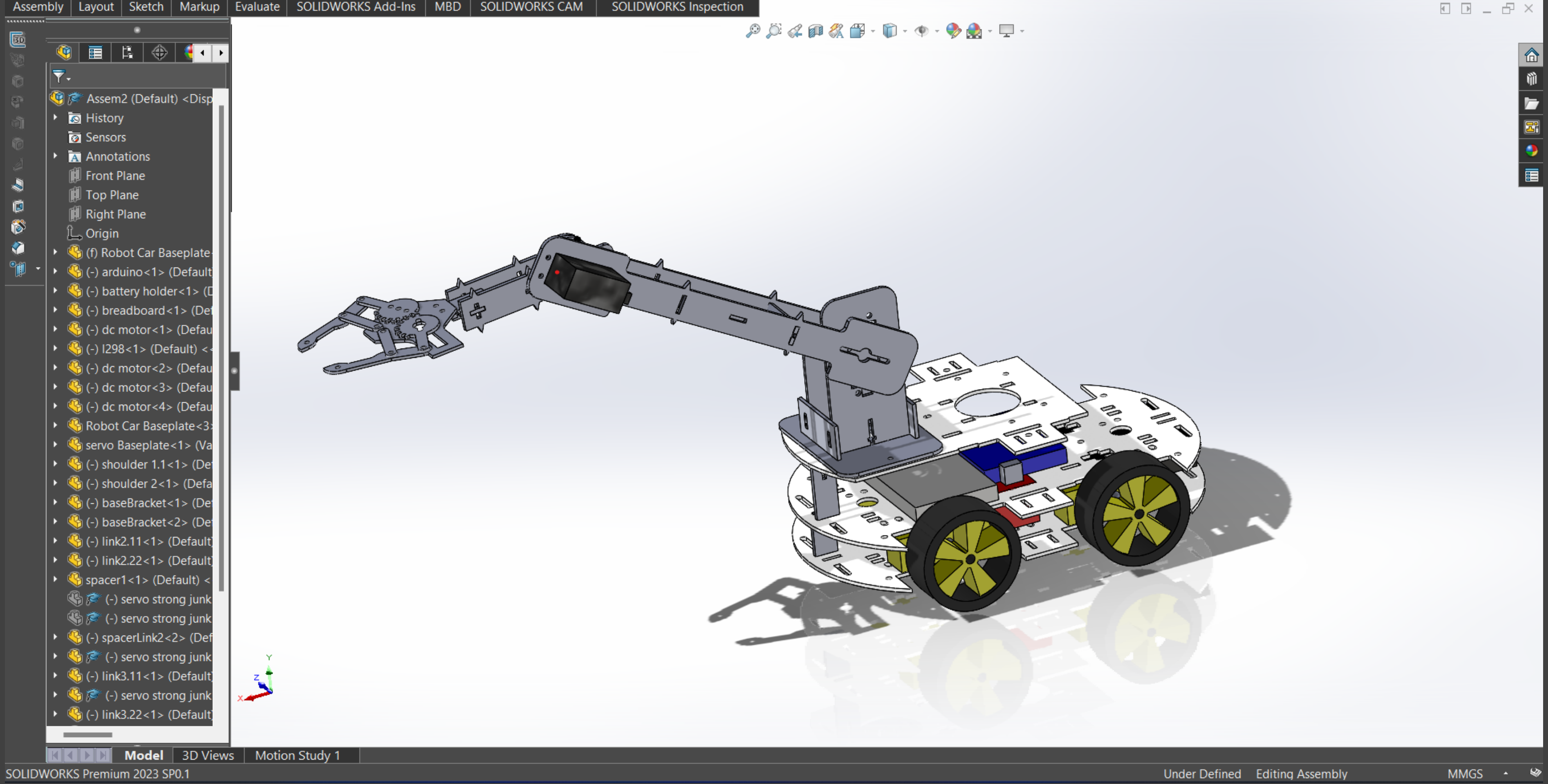
Eng. Hamed Abbas Abd El-Halim El-Sayyed

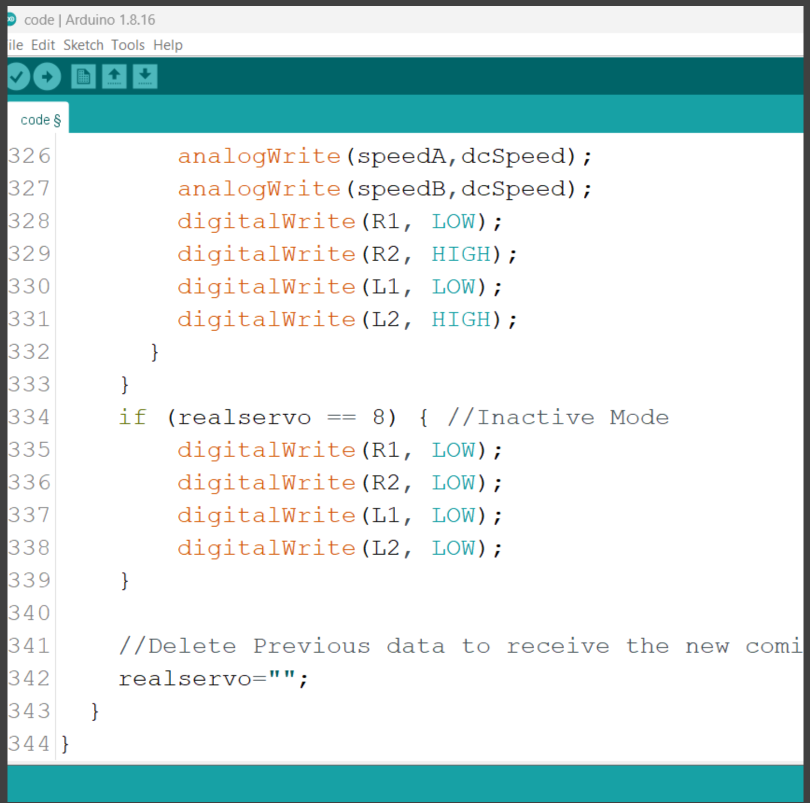
Product Design, R&D Mechanical Design Engineer

Bachelor of Mechatronics, Robotics, and Automation Engineering - Kafr El-Sheikh University, Egypt

hamed.abbas9922@gmail.com

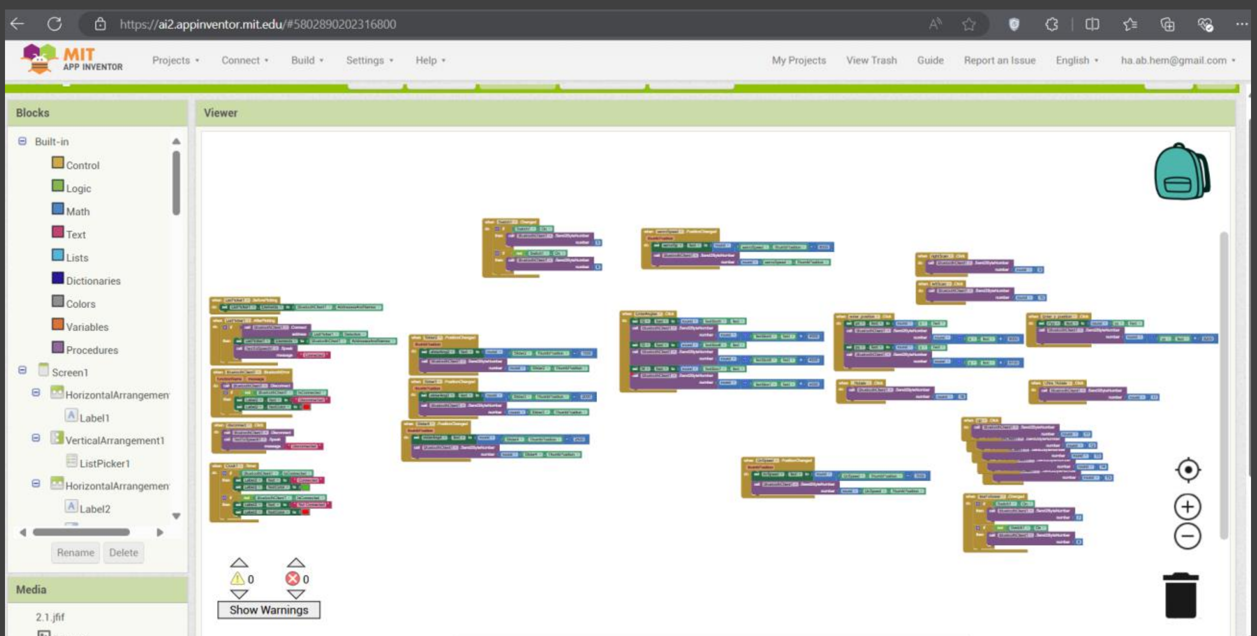
www.linkedin.com/in/hamed-abbas-a741ba176

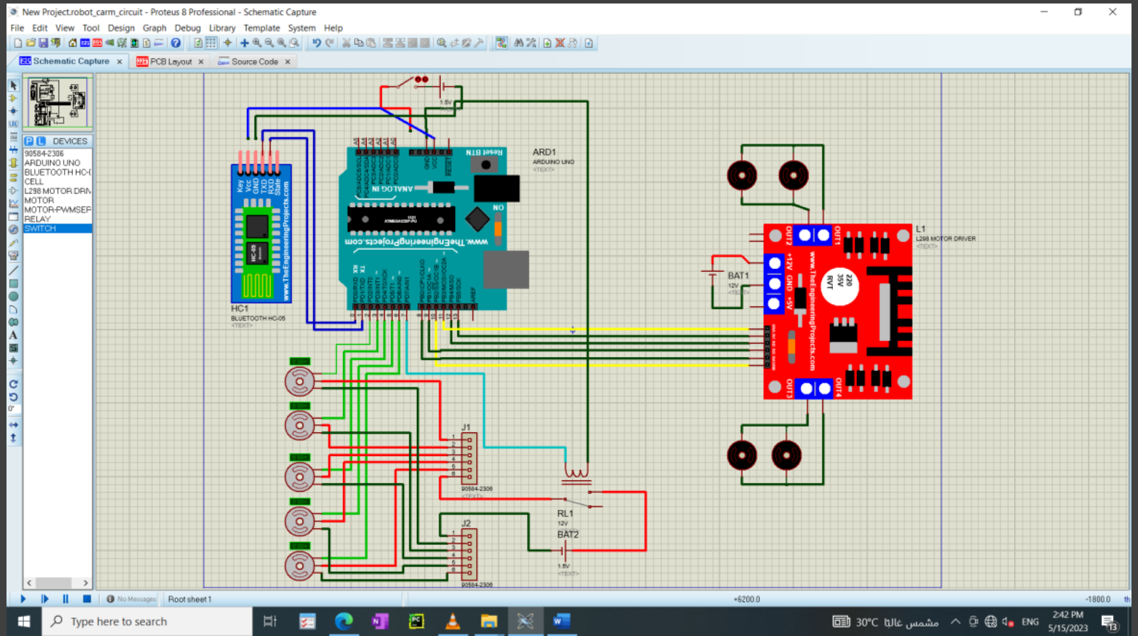




```
        analogWrite(speedA,dcSpeed);
        analogWrite(speedB,dcSpeed);
        digitalWrite(R1, LOW);
        digitalWrite(R2, HIGH);
        digitalWrite(L1, LOW);
        digitalWrite(L2, HIGH);
      }
    }
    if (realservo == 8) { //Inactive Mode
        digitalWrite(R1, LOW);
        digitalWrite(R2, LOW);
        digitalWrite(L1, LOW);
        digitalWrite(L2, LOW);
    }

    //Delete Previous data to receive the new comi
    realservo="";
  }
}
```





Chapter 1: Robotics

Robotics is a multidisciplinary field that involves the design, construction, operation, and use of robots to perform tasks traditionally done by humans. These tasks can range from simple assembly line operations to more complex activities like surgical procedures or space exploration.

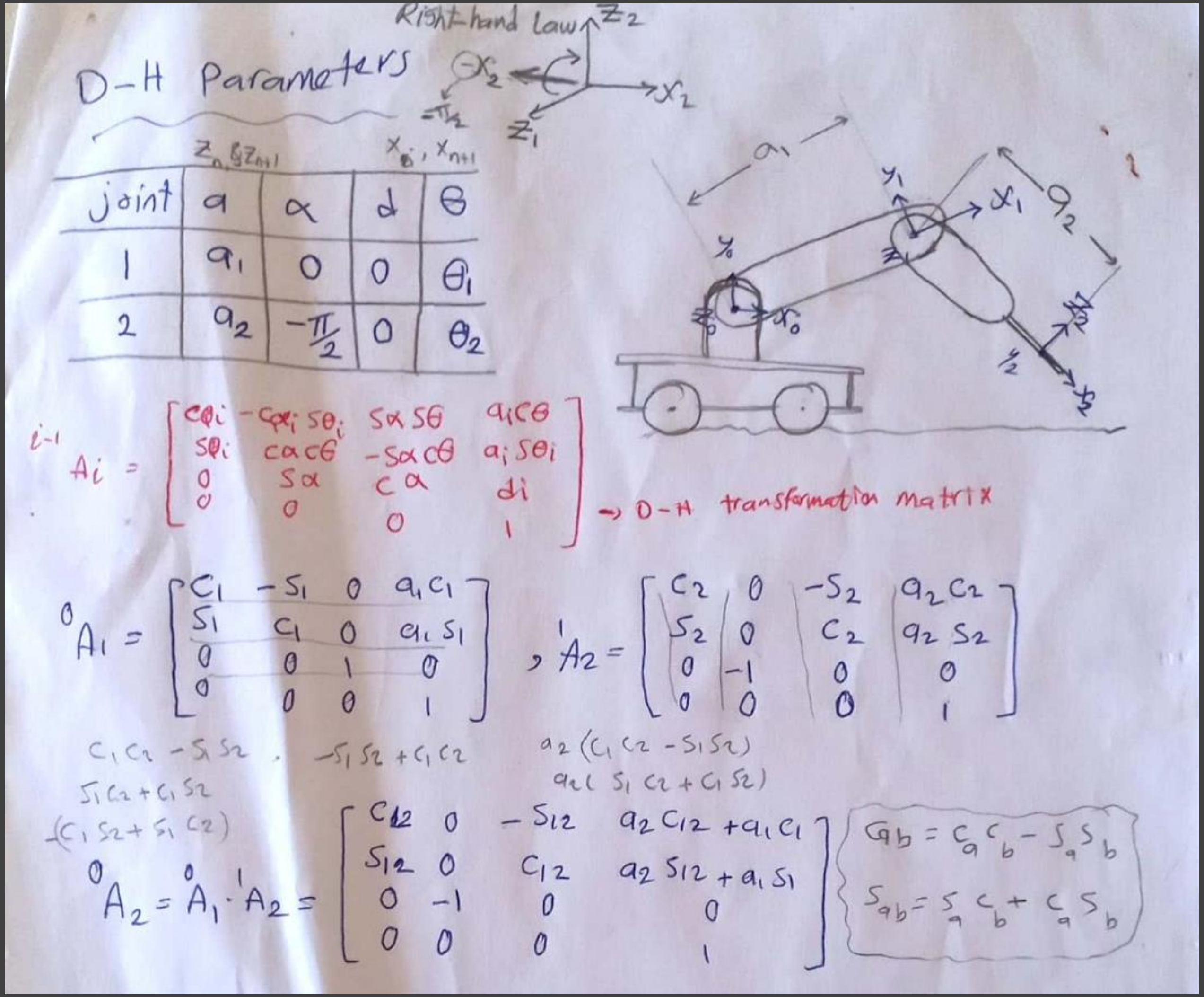
In the context of mobile robot manipulators, robotics plays a crucial role in enabling these machines to interact with their environment effectively. A mobile robot manipulator typically consists of two main components: a mobile base or car that provides the robot's mobility, and a robotic arm mounted on this base. The robotic arm is equipped with joints, links, and an end effector (such as a gripper) that enable it to manipulate objects in its environment.

The main function of a mobile robot manipulator is to pick up objects, transport them to a desired location, and place them there. This process involves a series of coordinated movements of the robotic arm and the mobile base to accurately navigate the robot to the target object, grasp it securely, and transport it to the specified destination.

By leveraging robotics in mobile robot manipulators, tasks that are repetitive, dangerous, or require precision can be automated, increasing efficiency, accuracy, and safety in various applications. These robots can be utilized in industries such as manufacturing, logistics, healthcare, and research, where they can perform tasks like material handling, inspection, maintenance, and more with high precision and reliability.

Robotics Calculations

In the case of our mobile robot manipulator, I have designed a system with a two-link robotic arm that includes two revolute joints and an end effector for gripping objects. By solving the robotics equations and deriving the inverse kinematics equations, I have obtained the necessary mathematical relationships to calculate the joint angles required to position the end effector at a specific coordinate (Px, Py).


Right-hand law
z_2
$-x_2$
x_2
$-\pi/2$
z_1
a_1
a_2
y_1
x_1
y_0
x_0
z_0
z_1
y_2
x_2

D-H Parameters

joint	a	α	d	θ
1	a_1	0	0	θ_1
2	a_2	$-\frac{\pi}{2}$	0	θ_2

$z_n \,\&\, z_{n+1}$ $\quad x_n, x_{n+1}$

$$^{i-1}A_i = \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha s\theta & a_i c\theta \\ s\theta_i & c\alpha c\theta & -s\alpha c\theta & a_i s\theta_i \\ 0 & s\alpha & c\alpha & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow \text{D-H transformation matrix}$$

$$^{0}A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1 c_1 \\ s_1 & c_1 & 0 & a_1 s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad ^{1}A_2 = \begin{bmatrix} c_2 & 0 & -s_2 & a_2 c_2 \\ s_2 & 0 & c_2 & a_2 s_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

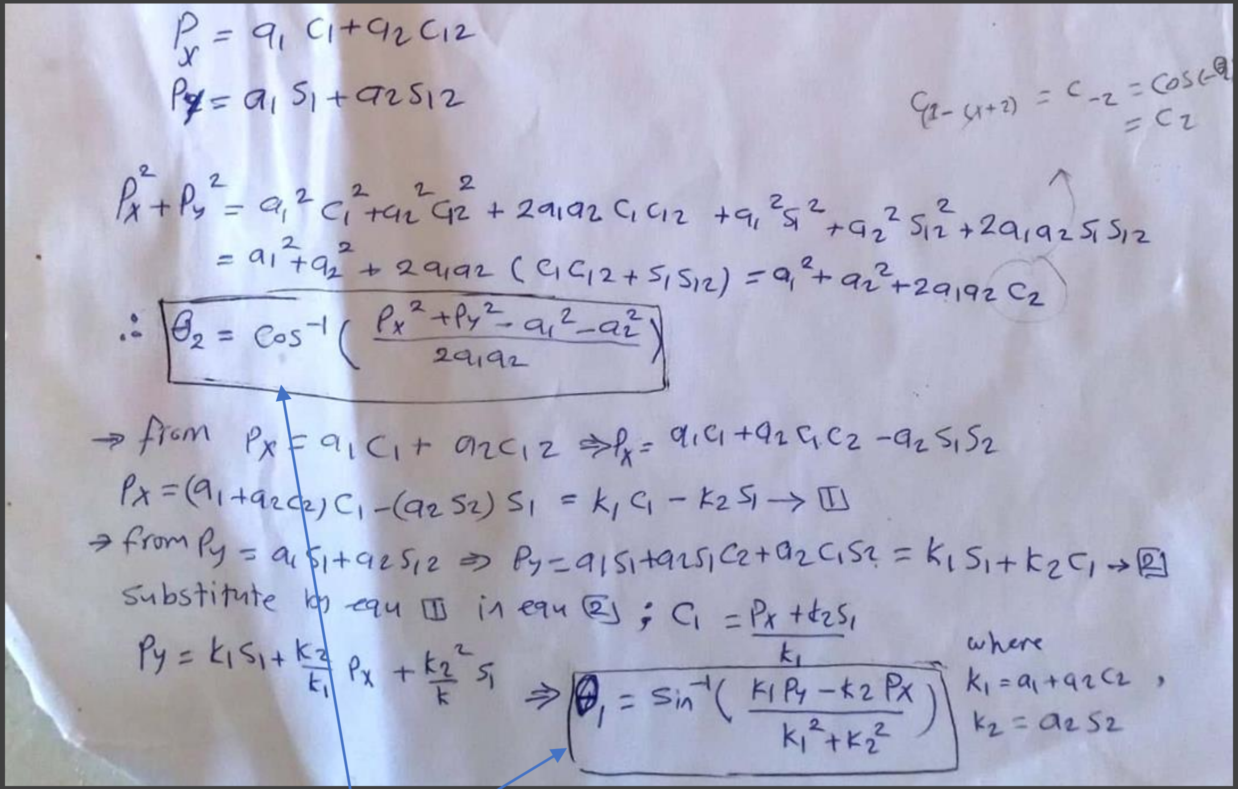
$c_1 c_2 - s_1 s_2$. $\quad -s_1 s_2 + c_1 c_2 \quad a_2(c_1 c_2 - s_1 s_2)$

$s_1 c_2 + c_1 s_2 \quad a_2(s_1 c_2 + c_1 s_2)$

$-(c_1 s_2 + s_1 c_2)$

$$^{0}A_2 = {}^{0}A_1 \cdot {}^{1}A_2 = \begin{bmatrix} c_{12} & 0 & -s_{12} & a_2 c_{12} + a_1 c_1 \\ s_{12} & 0 & c_{12} & a_2 s_{12} + a_1 s_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{\begin{aligned} c_{ab} &= c_a c_b - s_a s_b \\ s_{ab} &= s_a c_b + c_a s_b \end{aligned}}$$



$$P_x = a_1 c_1 + a_2 c_{12}$$
$$P_y = a_1 s_1 + a_2 s_{12}$$

$c_{1-(1+2)} = c_{-2} = \cos(-\theta_2) = c_2$

$$P_x^2 + P_y^2 = a_1^2 c_1^2 + a_2^2 c_{12}^2 + 2a_1 a_2 c_1 c_{12} + a_1^2 s_1^2 + a_2^2 s_{12}^2 + 2a_1 a_2 s_1 s_{12}$$
$$= a_1^2 + a_2^2 + 2a_1 a_2 (c_1 c_{12} + s_1 s_{12}) = a_1^2 + a_2^2 + 2a_1 a_2 c_2$$

$$\therefore \boxed{\theta_2 = \cos^{-1}\left(\frac{P_x^2 + P_y^2 - a_1^2 - a_2^2}{2a_1 a_2}\right)}$$

$\Rightarrow$ from $P_x = a_1 c_1 + a_2 c_{12} \Rightarrow P_x = a_1 c_1 + a_2 c_1 c_2 - a_2 s_1 s_2$

$P_x = (a_1 + a_2 c_2) c_1 - (a_2 s_2) s_1 = k_1 c_1 - k_2 s_1 \rightarrow \boxed{1}$

$\Rightarrow$ from $P_y = a_1 s_1 + a_2 s_{12} \Rightarrow P_y = a_1 s_1 + a_2 s_1 c_2 + a_2 c_1 s_2 = k_1 s_1 + k_2 c_1 \rightarrow \boxed{2}$

substitute ~~by~~ equ $\boxed{1}$ in equ $\boxed{2}$; $c_1 = \dfrac{P_x + k_2 s_1}{k_1}$

$P_y = k_1 s_1 + \dfrac{k_2}{k_1} P_x + \dfrac{k_2^2}{k} s_1 \Rightarrow \boxed{\theta_1 = \sin^{-1}\left(\dfrac{k_1 P_y - k_2 P_x}{k_1^2 + k_2^2}\right)}$

where $k_1 = a_1 + a_2 c_2$, $k_2 = a_2 s_2$

These are the equations which calculates joint angles needed to reach specific position (Px , Py)

a1: link_1 length
a2: link_2 length
Px and Py are the position coordinates in X and Y directions we need to manipulate from the robot origin

$$\boxed{\theta_1 = \sin^{-1}\left(\frac{k_1 P_y - k_2 P_x}{k_1^2 + k_2^2}\right)}$$

$$\boxed{\theta_2 = \cos^{-1}\left(\frac{P_x^2 + P_y^2 - a_1^2 - a_2^2}{2a_1a_2}\right)}$$

k1 = a1 + a2 * cos(theta_2)
K2 = a2 * sin(theta_2)
. First, theta_2 is calculated. Theta_1 is calculated using theta_2 value

These Equations are transformed into Arduino code to calculate the joint angles needed to reach a specific position

*Note that the Arduino trigonometric functions syntax differs from the mathematical syntax.
To calculate the sin, cos, and inverse cos,
You will need to use this syntax

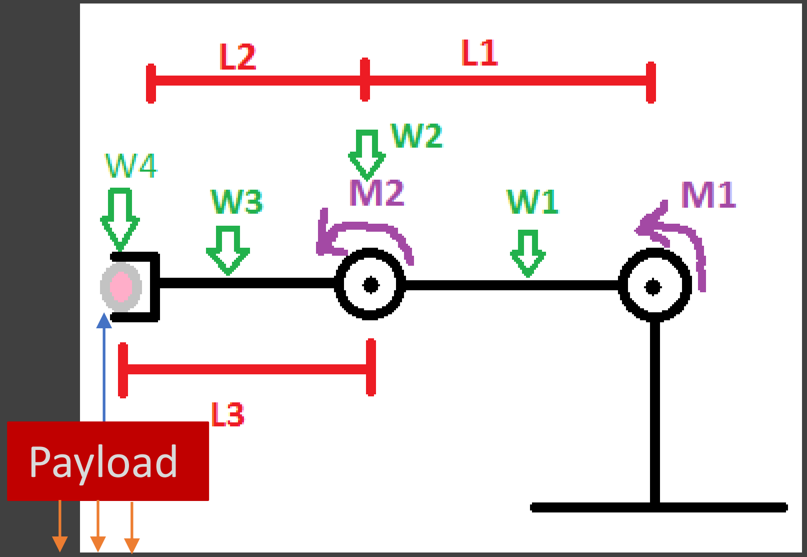
Arduino Trigonometric functions

* $\cos^{-1} \longrightarrow$ acos
* $\cos(\theta) \longrightarrow \cos\left(\frac{\theta * 3.14}{180}\right)$
* $\sin(\theta) \longrightarrow \sin\left(\frac{\theta * 3.14}{180}\right)$

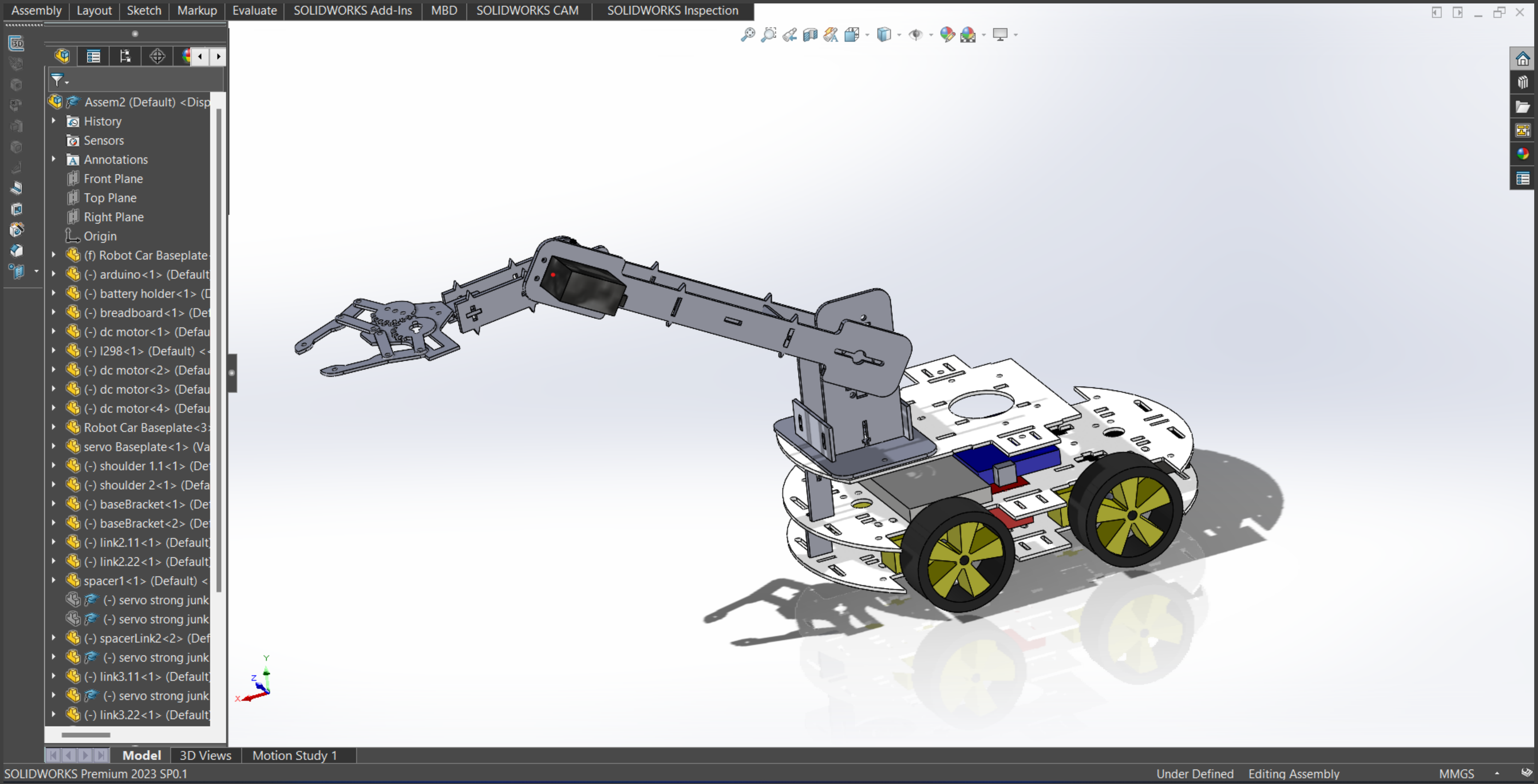
Designing the robotic arm with shorter links, denoted as $a1$ and $a2$, is recommended for several reasons:

Torque Minimization: Shorter links generally require less torque to move, aiding in reducing the overall torque requirements of the system. This can lead to more efficient operation of the motors in the robotic arm joints.
Increased Weight Capacity: By minimizing the torque required in each link, the robotic arm can potentially carry objects with higher weights. This is because the reduced torque demand allows for better weight-bearing capacity and stability in the arm.
Efficiency and Stability: With lower torque requirements, the motors used in the robotic arm joints can operate more efficiently. This increased efficiency not only saves energy but also contributes to the stability of the robotic arm during operation.

Therefore, by designing the robotic arm with shorter links, you are optimizing the arm for improved weight capacity, motor efficiency, and overall stability, enhancing the performance of the mobile robot manipulator in tasks such as object manipulation, transportation, and placement.



Chapter 2: Mechanical Design



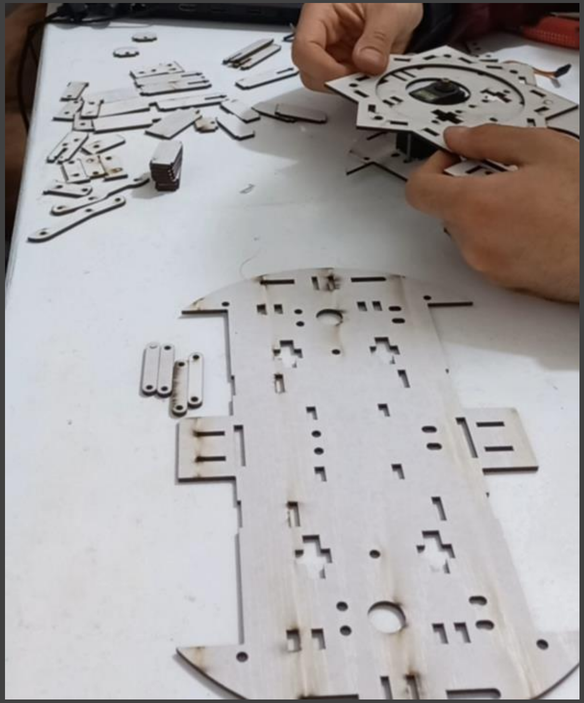
Assem2 (Default) <Disp
History
Sensors
Annotations
Front Plane
Top Plane
Right Plane
Origin
(f) Robot Car Baseplate
(-) arduino<1> (Default
(-) battery holder<1> (D
(-) breadboard<1> (Def
(-) dc motor<1> (Defau
(-) l298<1> (Default) <
(-) dc motor<2> (Defau
(-) dc motor<3> (Defau
(-) dc motor<4> (Defau
Robot Car Baseplate<3
servo Baseplate<1> (Va
(-) shoulder 1.1<1> (De
(-) shoulder 2<1> (Defa
(-) baseBracket<1> (De
(-) baseBracket<2> (De
(-) link2.11<1> (Default)
(-) link2.22<1> (Default)
spacer1<1> (Default) <
(-) servo strong junk
(-) servo strong junk
(-) spacerLink2<2> (Def
(-) servo strong junk
(-) link3.11<1> (Default)
(-) servo strong junk
(-) link3.22<1> (Default)

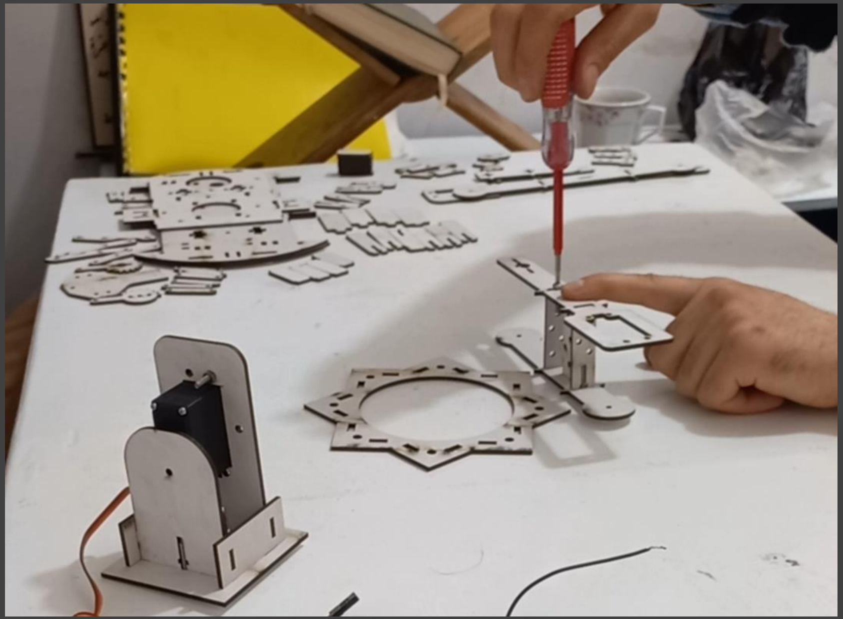
SOLIDWORKS Premium 2023 SP0.1   Under Defined   Editing Assembly   MMGS

[Mobile-Robot-Manipulator/Robot CARM mechanical design at main · HamedAbbas1420/Mobile-Robot-Manipulator · GitHub]

1. Body Design:

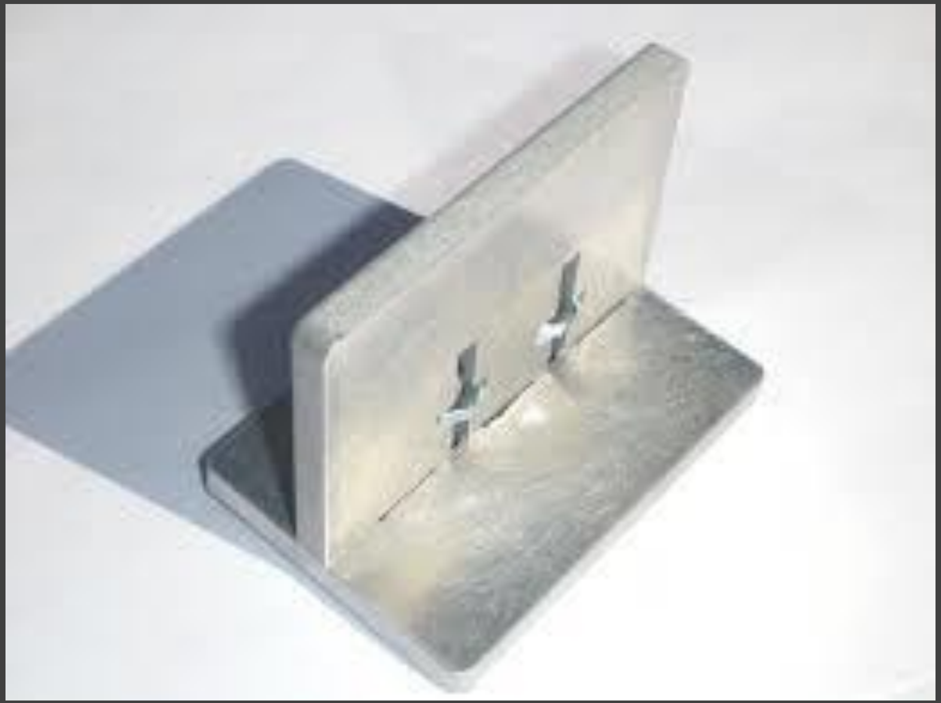
- The robot body is designed using 1.5mm Medium-density fiberboard (MDF) wood.
- The Tab and slot technique is utilized to connect the mechanical parts seamlessly.
- The slot holes are precisely sized to accommodate the acrylic thickness.
- Depending on the thickness used, customize the rectangular holes that fix related parts
- Glue or similar substances can be used to strengthen the connections between the parts.

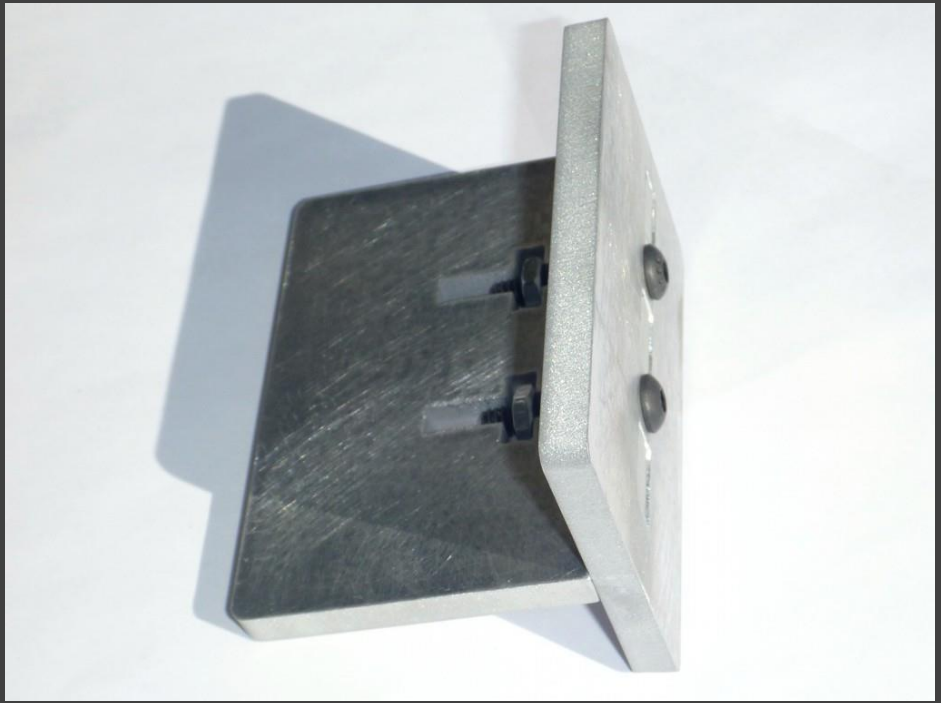

mdf chassis



How to use tab and slot plug technique?

.The mobile robot manipulator employs a tab and slot assembly technique for linking its components.
.This method involves creating a slot, achieved through an extrude cut, in one part,
and a tab, formed through an extrude boss feature in another part.
.The tab includes a central hole designed for accommodating a screw, facilitating secure fastening of the two components together.
.The central hole in the tab is customized depending on the screw and nut size and length



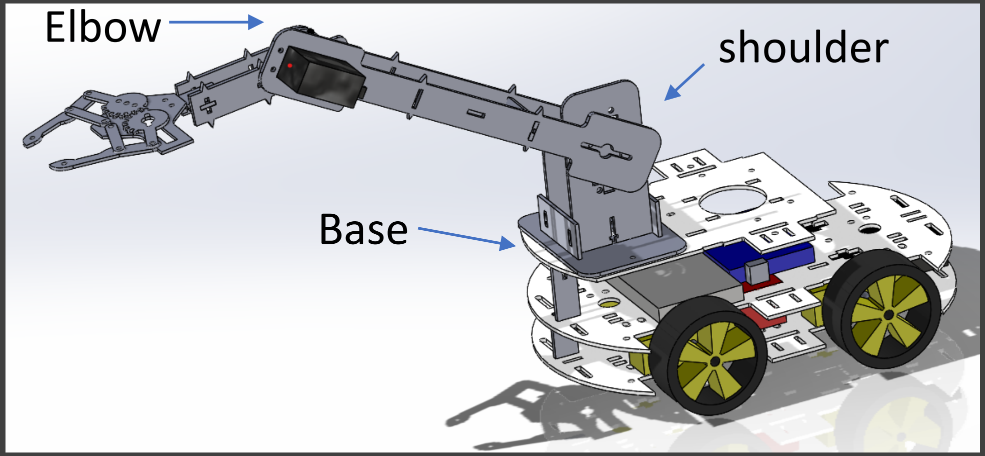


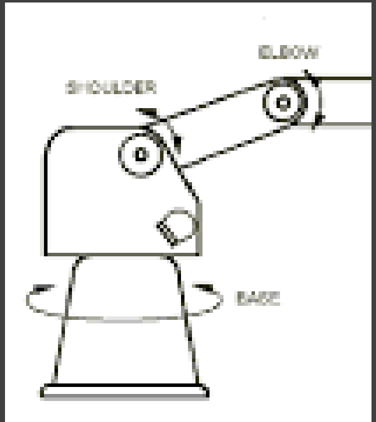
Mobile Robot Joints

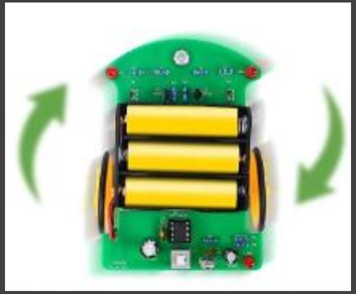
.The mobile robot manipulator is deigned with three revolute joints (Base, Shoulder, and Elbow)

.But in this project, I only activated two revolute joints (Shoulder and Elbow). WHY??

- To minimize electrical power; as the more servo motors used, the more current needed
- Easier assembly; the Base joint makes wiring complex, so I cancelled it. However, it is possible to activate if needed
- Less Functionality: The Base joint function can be replaced with the wheel turning. Base function is to make the robotic arm turn right and left, while the 4-wheel system do the same function.

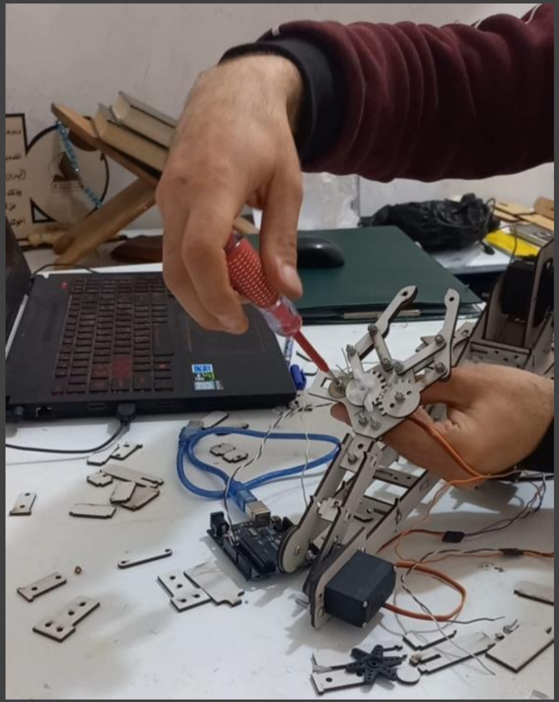






Mechanical Assembly:

- Open the SolidWorks file to know each part’s place and build the robot from the design
- Carefully align the acrylic pieces using the tab and slot plug method.
- Ensure a tight and stable connection between the various components, Secure them using 3mm diameter screws
- Apply glue or adhesive as needed to reinforce the assembly.
- use spacers to install the arm base to ensure free rotation
- Install each servo motor in its place as in the design and ensure its orientation
- Install the servo motor horn/arm on the appropriate Arm link

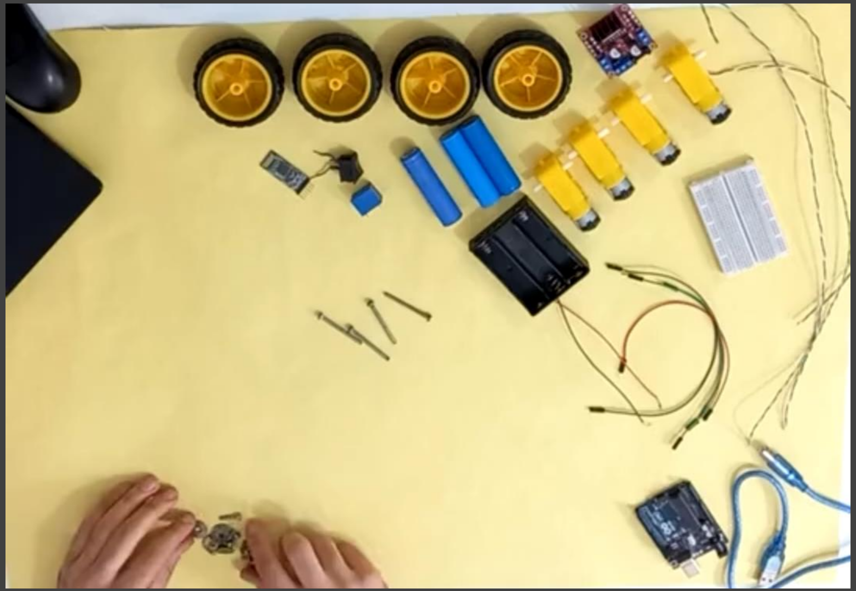


Chapter 3: Electrical Design and Electronics Integration

Building a mobile robot manipulator involves intricate electrical system design and meticulous wiring. Here is a comprehensive guide for creating the electrical system of the Mobile Robot Manipulator, focusing on the wheel system, servo motor utilization, and Bluetooth module integration.

Components:

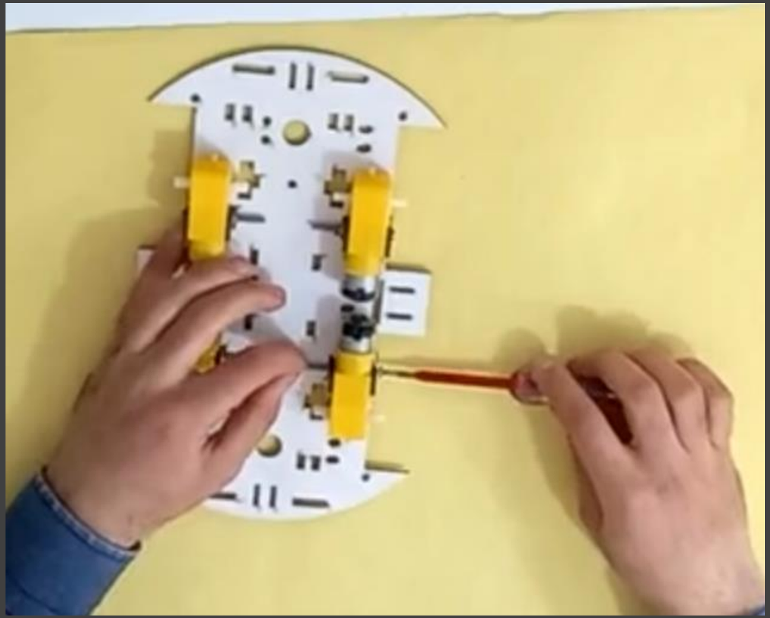
- Arduino Uno microcontroller
- Jumper wires (male-to-male, male-to-female, female-to-female)
- L298 H-bridge for motor control
- On/off switch
- 12V DC power source (3x Li-ion batteries at 3.7V each)
- 3-5V DC geared motors (4x for the wheels)
- Wheels (4x)
- MG995 servo motors (2x for the robotic arm)
- SG90 servo motor (1x for the gripper)
- 5V/5A DC power source for servo motors (Lipo batteries or use step-down buck converter)
- Relay to control the servo motor power source
- Bluetooth module (HC-05)

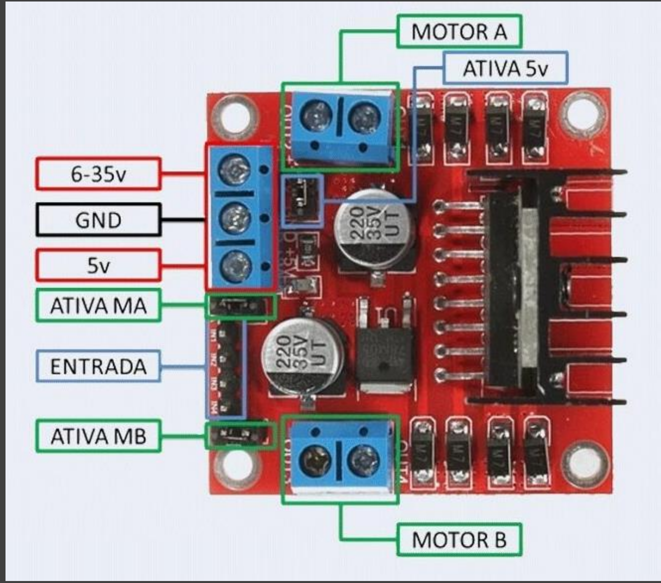


Circuit Wiring:

- **Wheel's DC Geared Motors**

. Connect the L298 H-bridge to the Arduino Uno pins for motor control
. Connect the right-side dc geared motors to the l298 motor A pin out
. Connect the left-side dc geared motors to the l298 motor B pin out
. Connect the h-bridge to a 12V dc power source:
  - You can use lipo batteries, OR
  - You can use 3x li batteries 3.7V connected in series
. Connect a ON/OFF switch between the battery and the h-bridge through the +ve wire; to easily Switch ON/OFF the Robot
. If you want to control the speed, use the ATIVA pin by connecting to Arduino pin





Arduino Uno power

Connect the 5v output of the L298 H-bridge to the Arduino Vin pin to power it

Perfecting Servo Motor Usage for the Robotic Arm:

Servo Motors:

- 2X MG995 servo motors for robotic arm joints
- 1X SG90 servo motor for gripper

Power Supply:

. Ensure safe operation within the servo motor's voltage range of 4.8-7.2 volts. Higher voltage may damage the motor

. Ensure sufficient current for servo motors, the Mg995 servo motor:

-No load operating current draw: 170mA, - Current at maximum load: 1200mA

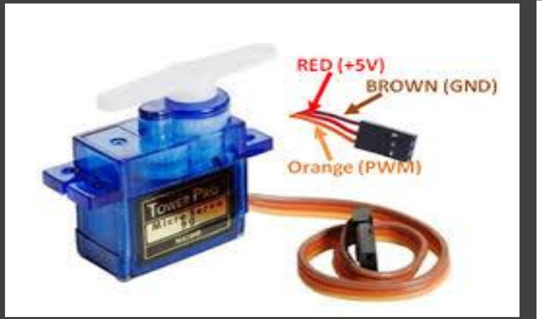
That means if you are using 2X mg995 servo motors and a sg90 motor,

you need about 4A,

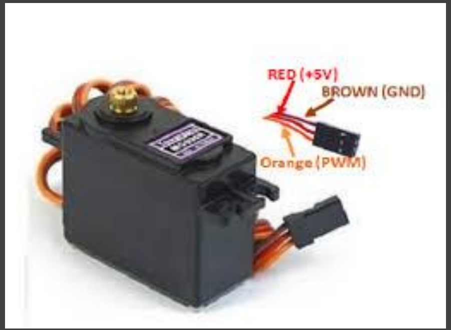
. Use 2X Li 3.7V batteries connected in series with a step-down buck converter module to regulate the voltage to the safe range

You can review the mg995 datasheet for further info.

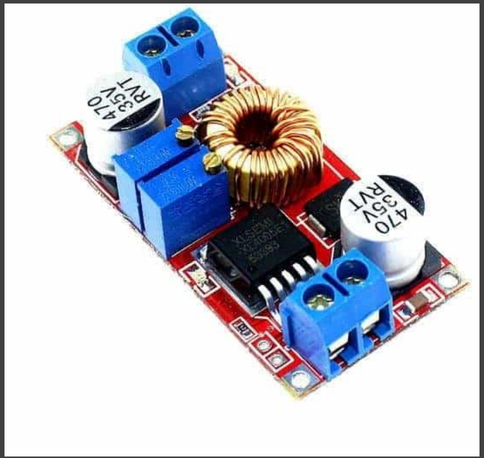
. Use a charging module (e.g. TP4056) when a battery gets empty

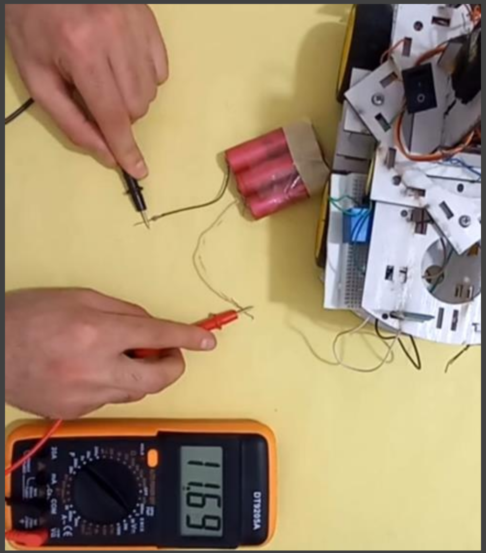


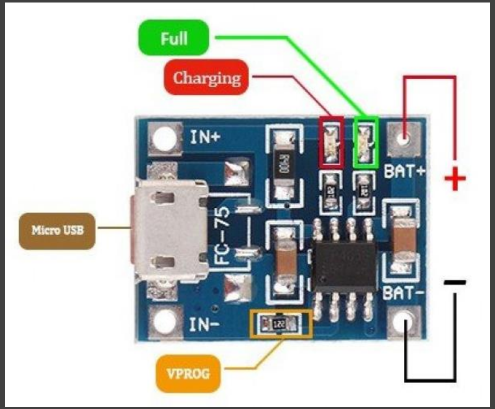
SG90 servo motor

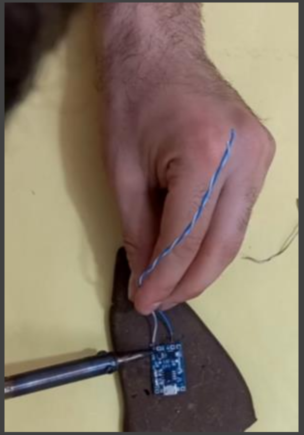


Mg995 servo motor


XL4015E1 5A DC-DC buck step-down







Noise Reduction:

Introduce a capacitor and a resistor in the servo motor power circuit to dissipate noise, ensuring smooth and stable performance. Experiment with values for optimal noise suppression.

Power Control:

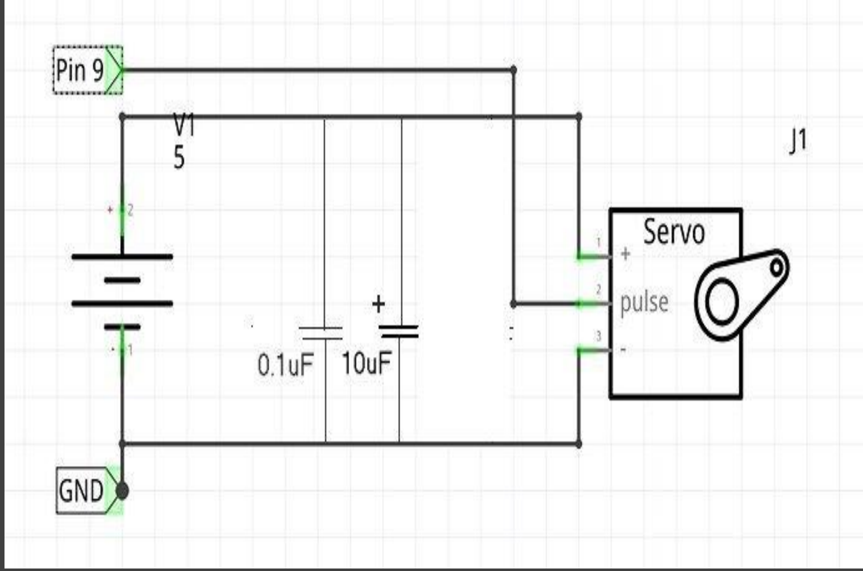
Implement a relay to manage the power source, allowing for convenient on/off control of the circuit.

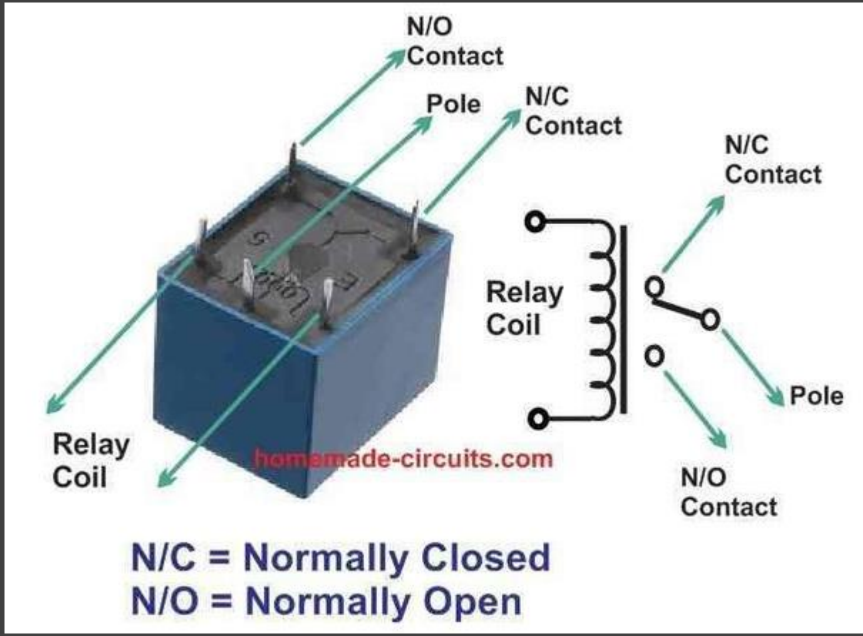
Power Indication:

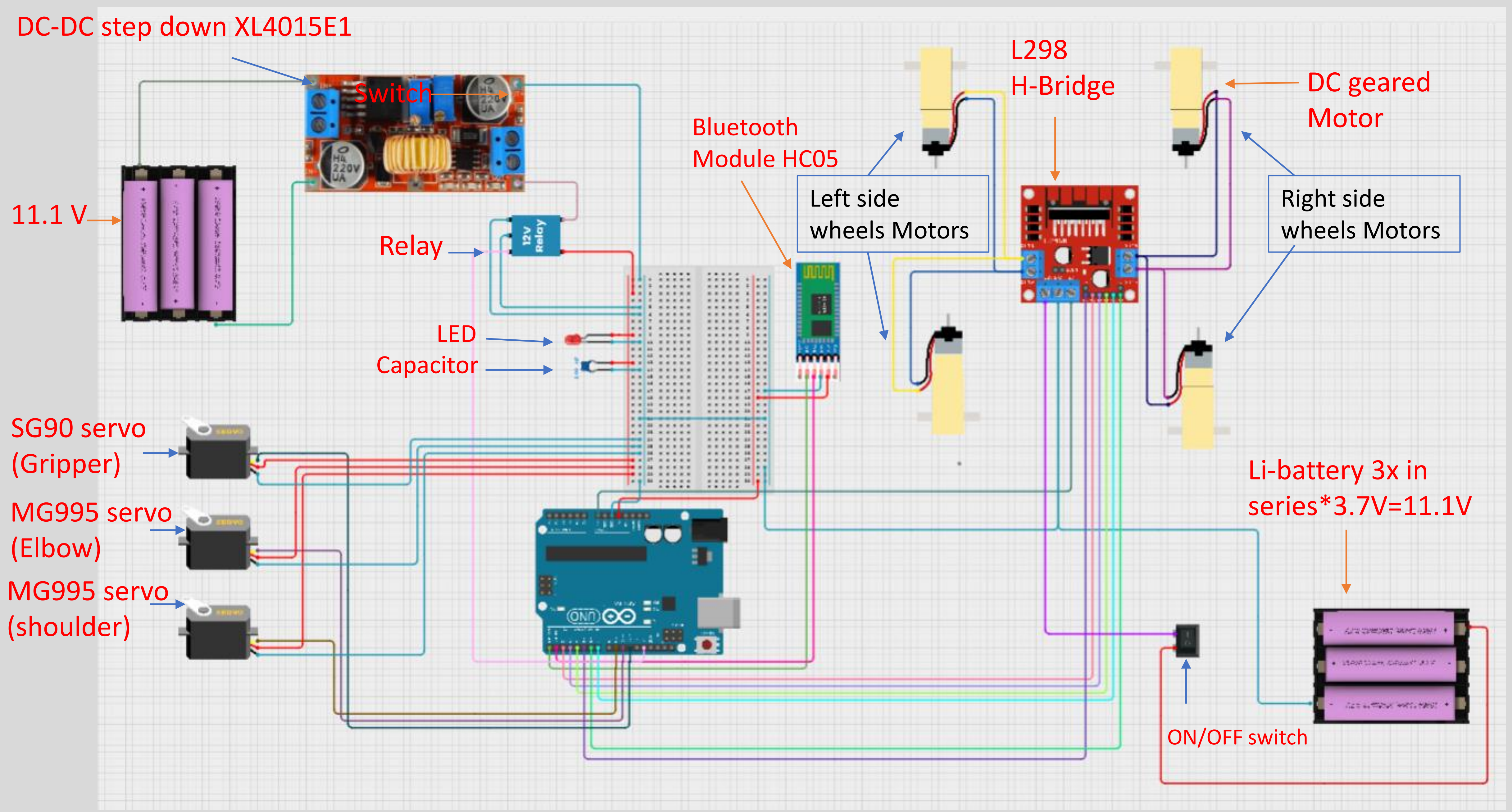
Add a LED to the servo motor circuit that indicates whether the Power is ON or OFF

Grounding for Enhanced Performance:

To ensure stable operation and prevent electrical noise, unify the grounds of all components.
Connect the grounds of the L298 H-bridge, servo motors, Bluetooth module, power sources, and all circuit elements together.



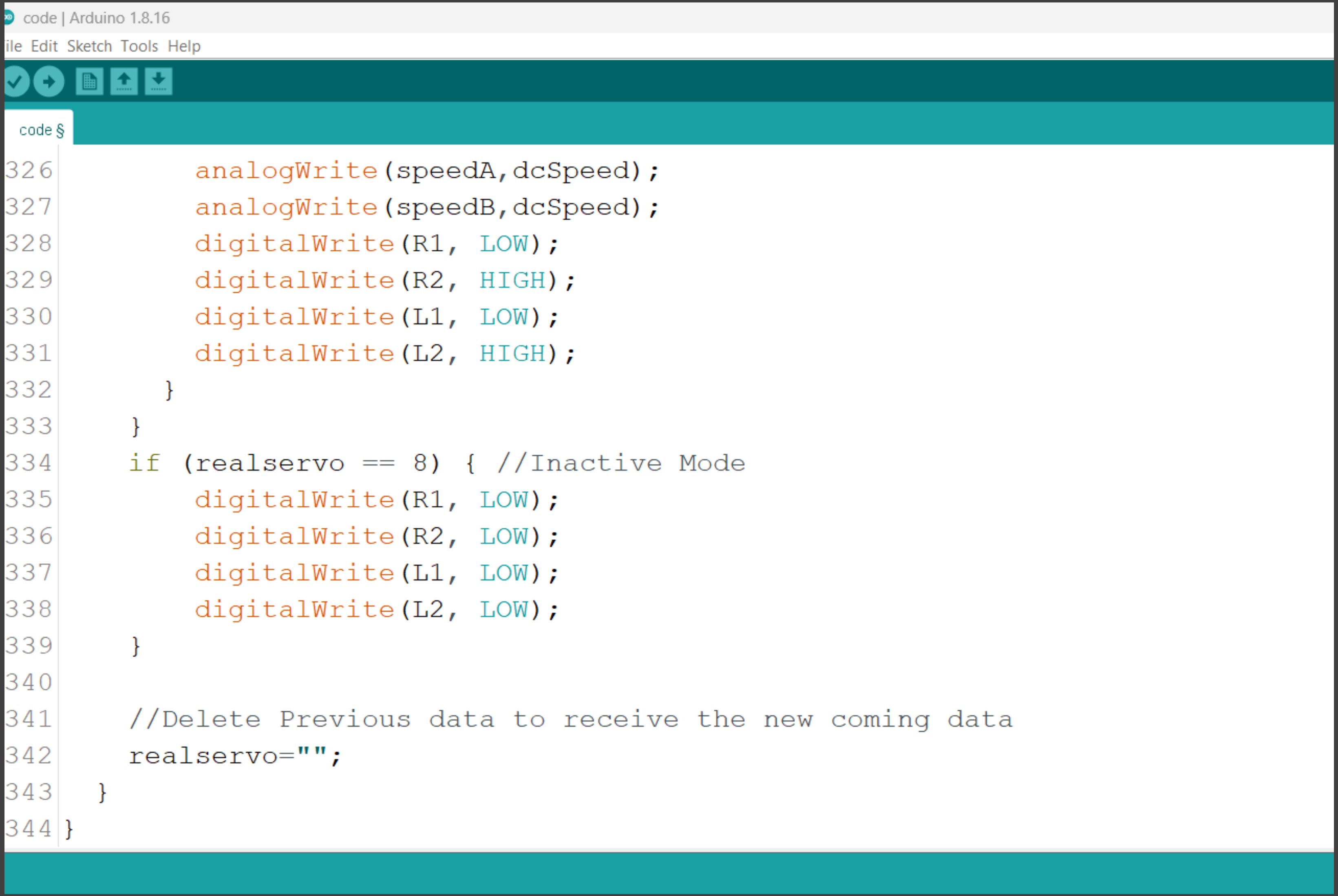




Wiring and circuit diagram

https://app.cirkitdesigner.com/project/589feb54-0bd7-4a68-a908-65638cb6498b

Chapter 4: Software (Arduino coding)



```
          analogWrite(speedA,dcSpeed);
          analogWrite(speedB,dcSpeed);
          digitalWrite(R1, LOW);
          digitalWrite(R2, HIGH);
          digitalWrite(L1, LOW);
          digitalWrite(L2, HIGH);
        }
      }
      if (realservo == 8) { //Inactive Mode
          digitalWrite(R1, LOW);
          digitalWrite(R2, LOW);
          digitalWrite(L1, LOW);
          digitalWrite(L2, LOW);
      }

      //Delete Previous data to receive the new coming data
      realservo="";
    }
}
```

Mobile-Robot-Manipulator/Software (Arduino Coding)/CARM arduino/code/code.ino at main · HamedAbbas1420/Mobile-Robot-Manipulator · GitHub

Code Description:

Libraries: The code utilizes the SoftwareSerial library for Bluetooth communication and VarSpeedServo library for controlling servo motor speeds.

Initialization: Pins for motors, servos, Bluetooth communication, and relay are set up in the setup() function.

Servo Setup: Three servos (shoulder, elbow, gripper) are attached and initialized to default positions.

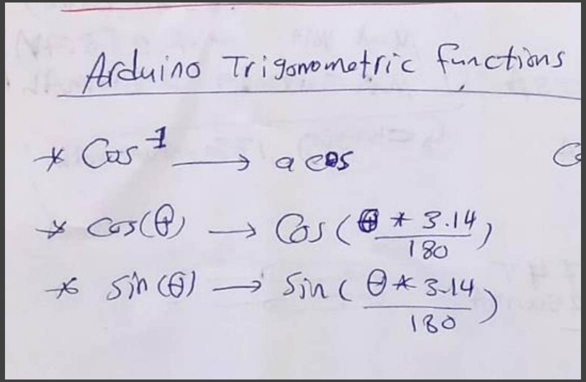
Bluetooth Communication: The code listens for Bluetooth commands to control servo positions, servo speeds, and trigger relay actions.

Inverse Kinematics: The key algorithm in the code is the conversion of inverse kinematics equations to control servo angles based on desired end effector positions.

Drive Control: The code also includes logic for controlling the L298 H-bridge to drive the four wheels of the robot.

Algorithms:

Inverse Kinematics Conversion:
The code calculates the angles required for the arm joints based on the desired end effector position (Px, Py) using inverse kinematics equations.
It computes the angles for the shoulder and elbow servos (theta_f1 and theta_f2) to achieve the desired position.
As we mentioned previously in chapter 1, Arduino trigonometric functions syntax differs from the mathematical syntax



Arduino Trigonometric functions

* $\cos^{-1} \longrightarrow$ acos
* $\cos(\theta) \longrightarrow \cos\left(\frac{\theta * 3.14}{180}\right)$
* $\sin(\theta) \longrightarrow \sin\left(\frac{\theta * 3.14}{180}\right)$

Bluetooth Control:
The code listens for Bluetooth commands to adjust servo positions, servo speeds, and trigger actions like turning the relay on or off.
Different ranges of Bluetooth input values are mapped to control different components of the robot arm.

Servo Speed Control:
The servo speed is adjusted based on the input values received over Bluetooth, allowing for smooth and controlled movements of the robot arm.

Wheel System Control

DC Motor Speed Control:

The code reads the value of realservo to determine the speed at which the DC motors should operate. The speed value is captured within a specific range (7000 to 7255) and stored in the variable dcSpeed.

Movement Commands:

The code interprets different values of realservo to determine the direction of movement for the mobile robot.

Move Forward (Command 11)

Move Reverse (Command 13)

Turn Left (Command 15)

Turn Right (Command 14)

Stop (Command 12)

What are these command numbers?!

The Arduino code described interfaces with a custom MIT App Inventor application designed for remote control of the mobile robot via Bluetooth communication. The MIT App Inventor application features an intuitive interface comprising buttons and text boxes. Each button on the interface is programmed to transmit a specific numerical value representing a command to the robot through the established Bluetooth connection.

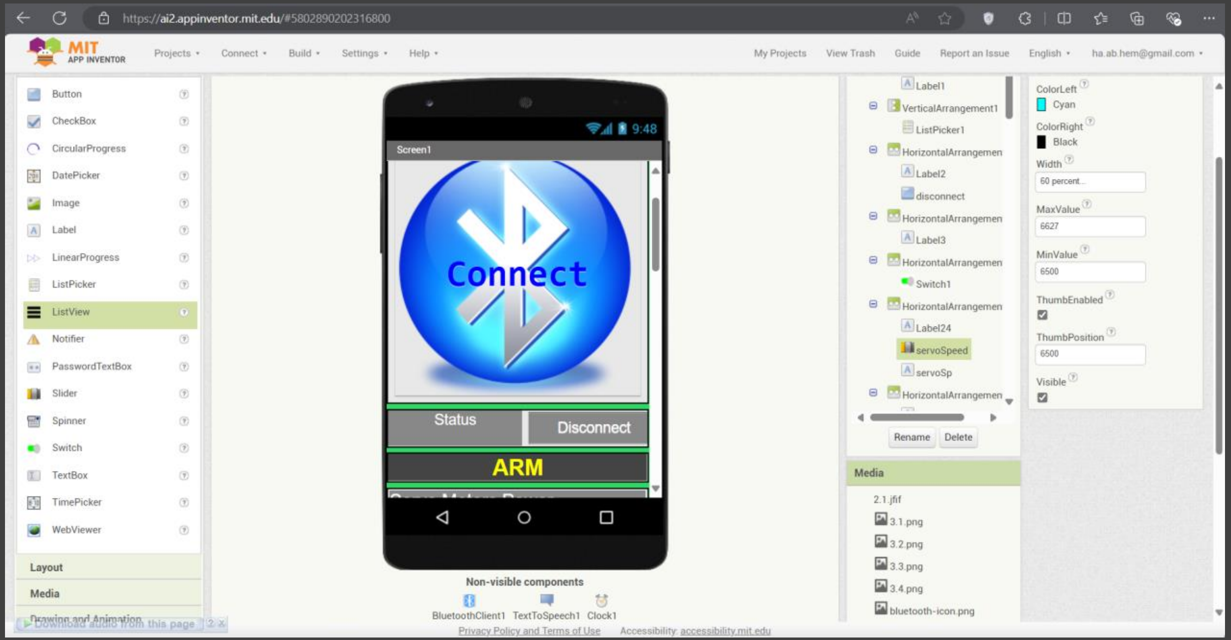
The numerical values sent from the MIT App Inventor application correspond to predefined commands that dictate the robot's behavior and movements. These commands are interpreted by the Arduino code, enabling the robot to execute actions such as rotating a servo with specific value, sending inverse kin. Parameters values, moving forward, reversing, turning left, turning right, and stopping based on the received values.

This setup allows for seamless and user-friendly remote control of the mobile robot, where users can interact with the MIT App Inventor interface to send commands wirelessly to the robot via Bluetooth, facilitating intuitive and efficient operation of the robot's locomotion and maneuvering capabilities.

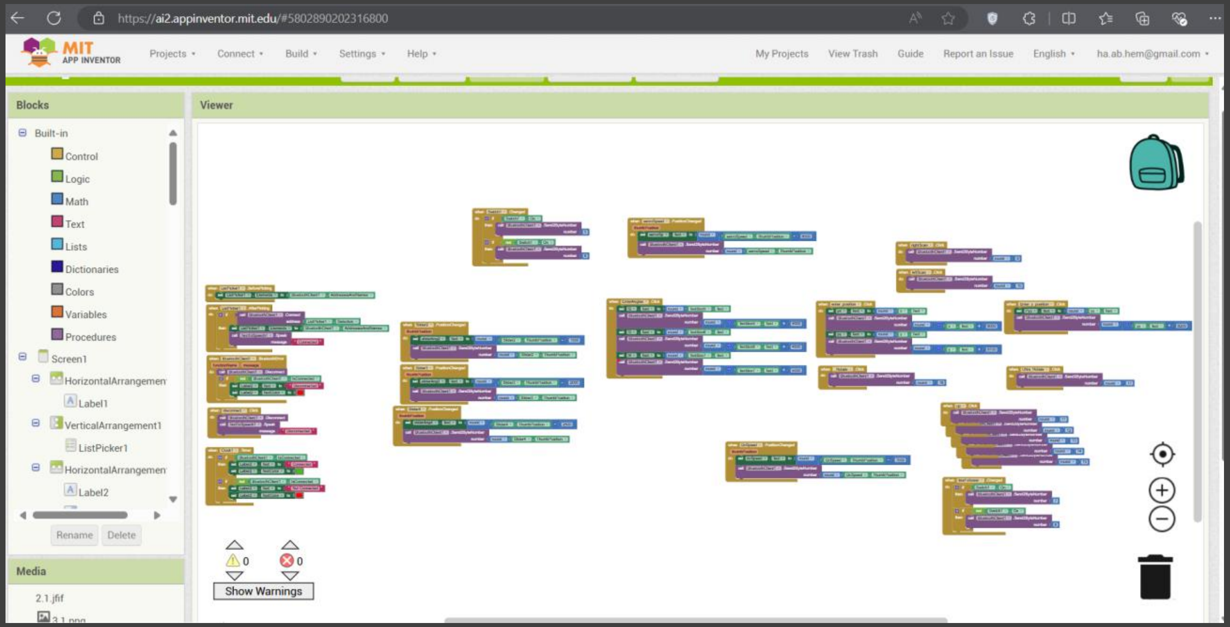
Chapter 5: Interface (MIT App Inventor Application)

The MIT App Inventor platform is a powerful tool that enables users to create custom mobile applications with ease, even without any extensive programming experience. The platform supports the development of applications for both Android and iOS devices. In this context, we will discuss how to install, configure, and utilize a specific MIT App Inventor-generated APK for remote controlling a mobile robot manipulator.

[Mobile-Robot-Manipulator/MIT app inventor CARM app at main · HamedAbbas1420/Mobile-Robot-Manipulator · GitHub]()

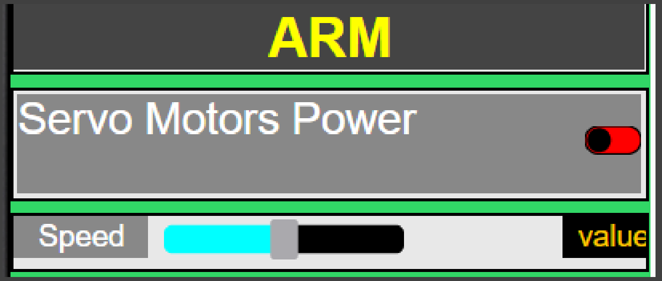


ColorLeft: Cyan
ColorRight: Black
Width: 60 percent...
MaxValue: 6627
MinValue: 6500
ThumbEnabled: ✓
ThumbPosition: 6500
Visible: ✓

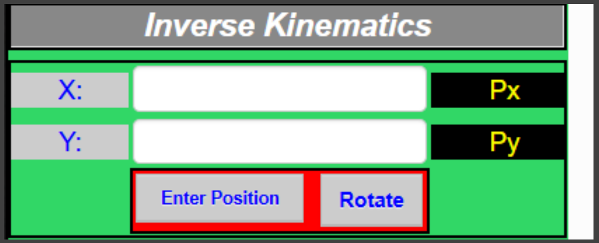


App Interface

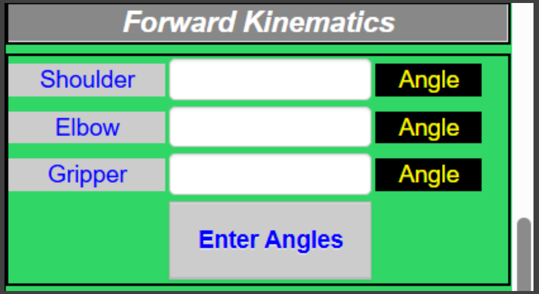
Servo motors power: Switch ON/OFF relay of servo motors power source
Speed: User defines servo motors speed using the slider



Inverse Kinematics:
. User enters the desired position for the end effector in the text box
. Enter position: to send values via Bluetooth to the microcontroller
. Rotate: Apply values sent to the robotic arm

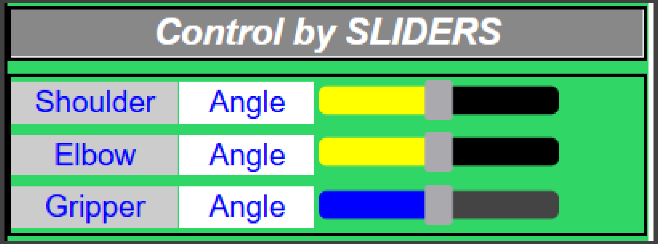


Forward Kinematics:
. User enters the desired angles in the text box
. Enter Angles: to send values via Bluetooth to the microcontroller and rotate the robotic arm

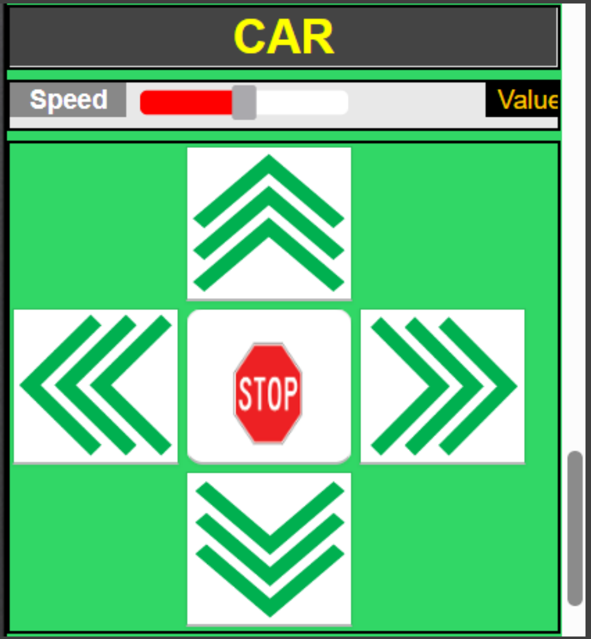


App Interface

Control by sliders:
It enables the user to control the joint via a slider for easy manipulation

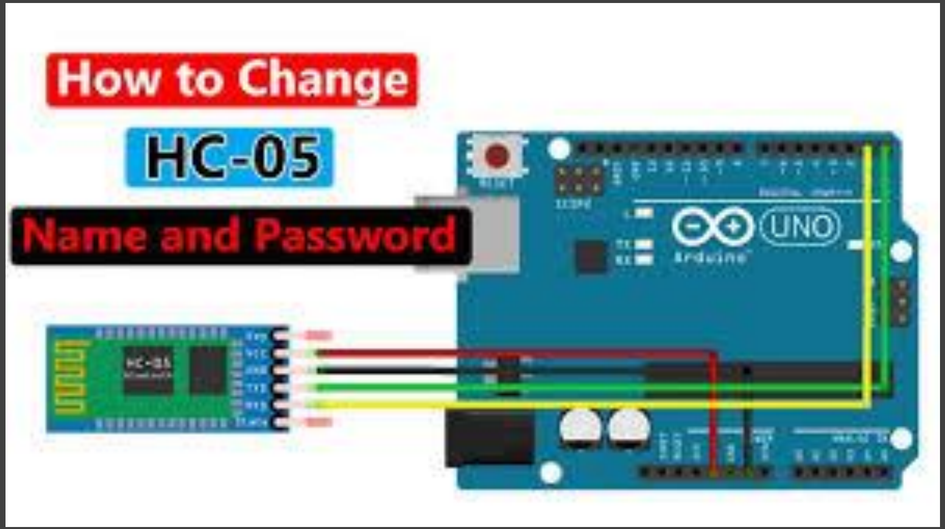


Car/ Wheels control:
Speed: DC motor speed can be controlled with a value range of 0~255
Pulse width modulation
Directions: One click makes wheels rotate to the proper direction



Customizing the Bluetooth Module:

To streamline the usage of the Bluetooth module, research online methods to modify its settings for improved functionality and ease of use. By making appropriate adjustments to the Bluetooth module, you can enhance the overall user experience when interacting with the mobile robot via the MIT App Inventor application.

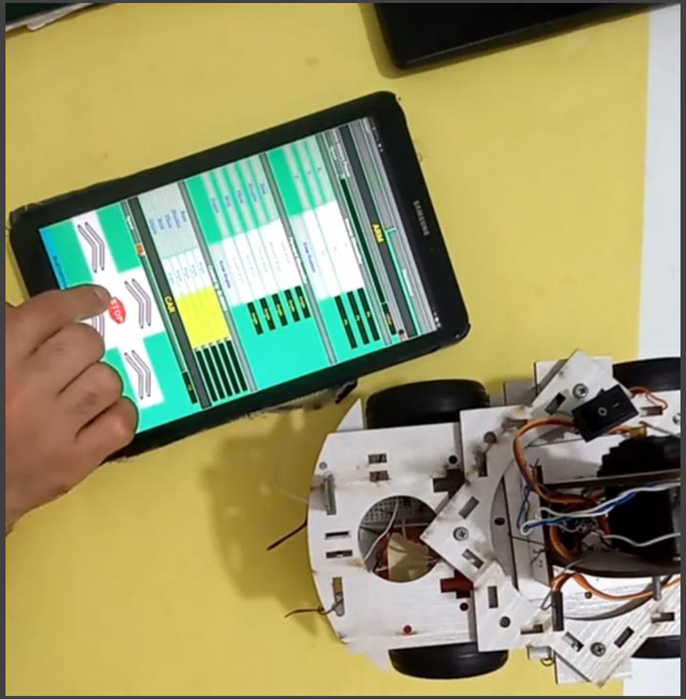


Utilizing the MIT App Inventor Application:

You can access the APK file from the designated repository on GitHub, enabling you to download and install the application on your Android device. It's important to note that MIT App Inventor applications are compatible with earlier versions of Android (e.g., Android 6, 7, 8, 9) and may not function optimally on older versions.

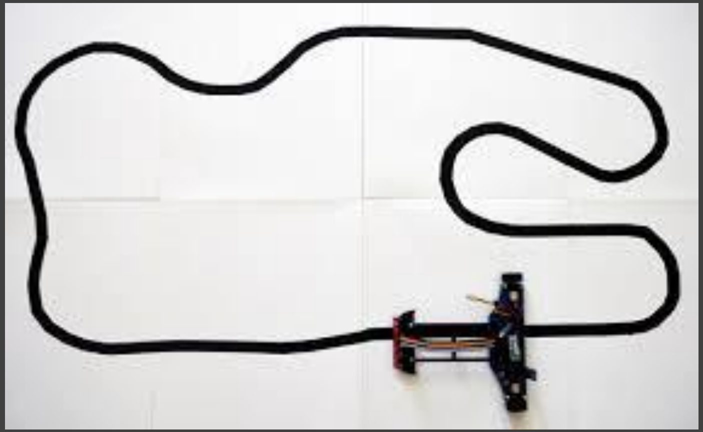
Uploading the Arduino Code:
Ensure the Arduino code (uploaded on my github repo) for controlling the robot is uploaded and integrated with the MIT App Inventor application. This code serves as the backbone for interpreting commands sent from the application and executing corresponding actions on the robot.

Installation and Connection Process:
Upon installing the MIT App Inventor application on your device, activate the robot and establish a connection with the Bluetooth module. By following these steps, you can seamlessly interact with the robot using the customized MIT App Inventor application, leveraging its intuitive interface to control the robot's movements and functionalities effectively.



Ideas for upgrading the mobile robot manipulator and adding new features

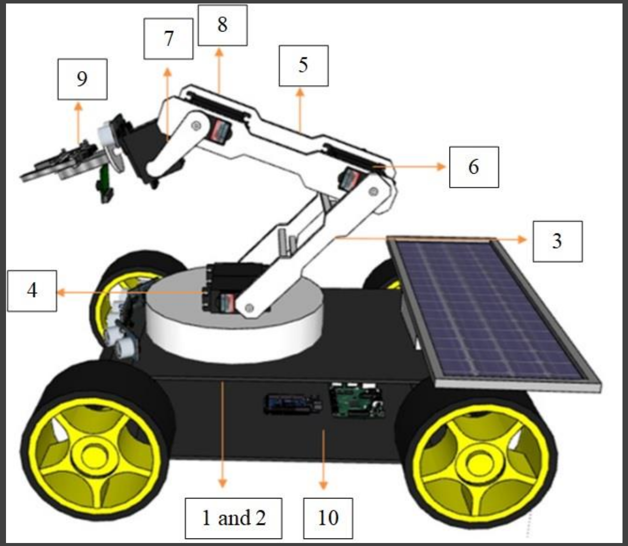
Line Follower System:
Incorporate a line follower sensor to enable the robot to follow predefined paths. This feature enhances efficiency by allowing the robot to navigate autonomously along specified routes, facilitating movement from the origin station to the target station.



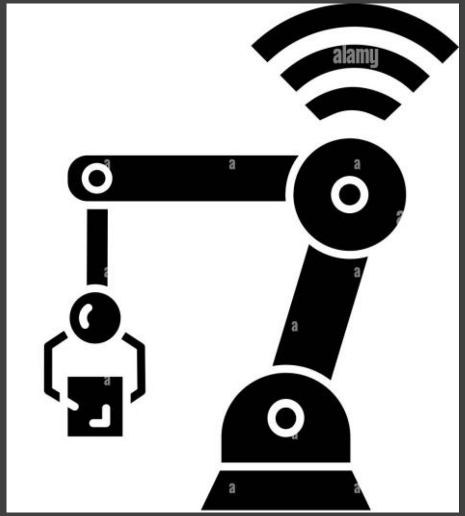
Computer Vision and AI:
Equip the robot with a camera and computer vision system powered by AI. This upgrade enhances the robot's performance by enabling it to:
Calculate inverse kinematics autonomously for precise positioning.
Identify and avoid obstacles in its path.
Interact with the workstation environment more effectively, improving overall operational efficiency.

Ideas for upgrading the mobile robot manipulator and adding new features

Solar Power Integration:
Integrate a solar cell system into the robot to serve as an alternative power source. This addition not only reduces reliance on traditional electricity but also promotes sustainability and environmental friendliness by harnessing renewable energy.



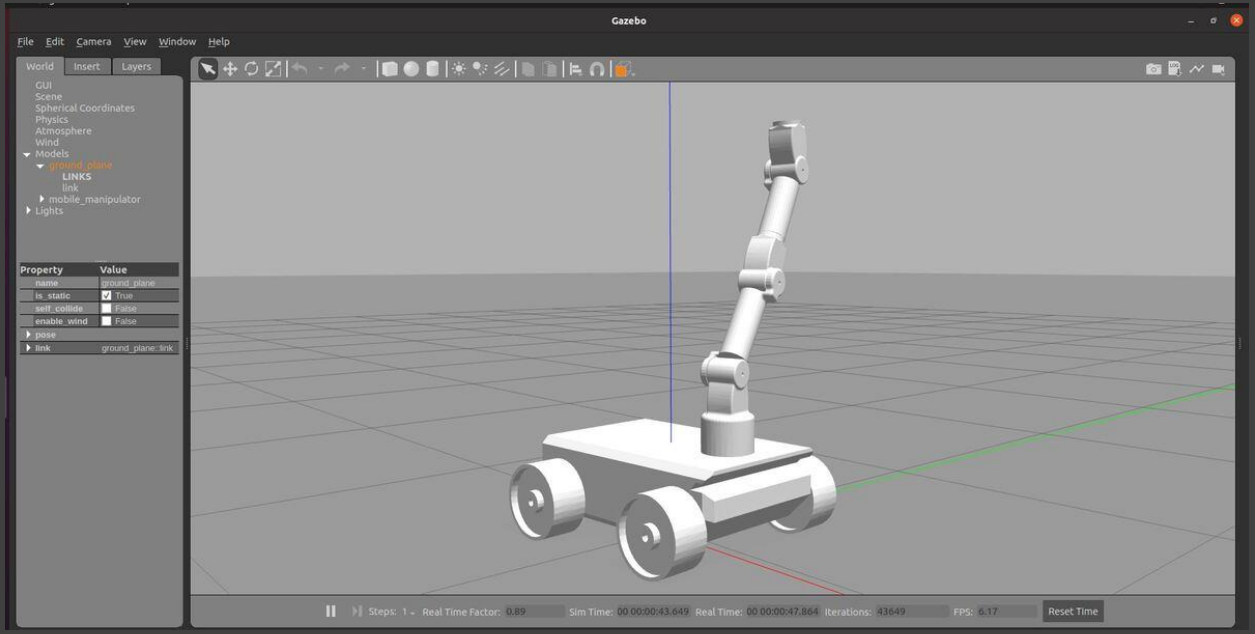
IoT Connectivity:
Establish a connection between the robot and an IoT system to enhance user-friendliness and remote-control capabilities. By linking the robot to IoT, users can:
Control the robot remotely from a centralized station.
Monitor and manage the robot's operations, status, and performance in real-time.
Receive alerts and notifications regarding the robot's activities and maintenance requirements.



ROS (Robot Operating System)

ROS stands for Robot Operating System. It is an open-source middleware framework widely used in the robotics community to develop and control robotic systems. Despite its name, ROS is not an operating system but rather a collection of software frameworks for robot software development.
Key features of ROS include:

Node-based architecture: ROS uses a peer-to-peer architecture where individual components (nodes) communicate with each other via messages sent over topics.
Message passing: Nodes communicate with each other by publishing messages to topics, allowing for asynchronous communication between different parts of a robotic system.
Package management: ROS uses a package-based system to organize and distribute code and resources. Users can easily share and reuse code through ROS packages.
Tools: ROS provides a range of tools for visualization, simulation, debugging, and monitoring of robotic systems. Some popular tools include RViz for visualization and Gazebo for simulation.
Community support: ROS has a large and active community of developers contributing to the ecosystem, which results in a wide range of libraries, frameworks, and resources being available for robotic development.
Platform independence: ROS is designed to be platform-independent, allowing it to run on various operating systems such as Linux, macOS, and Windows.



Advantages of Using ROS in Mobile Robot Manipulators:

Modularity and Scalability: ROS enables modular development, making it easier to integrate different components of the robot system. This modularity allows for scalability as the robot's capabilities expand.
Advanced Control and Navigation: ROS offers advanced control algorithms, navigation packages, and simulation tools that can enhance the robot's autonomy and efficiency in tasks such as mapping, localization, and path planning.
Sensor Integration: ROS provides extensive support for various sensors, allowing for seamless integration of sensor data into the robot's decision-making processes.
Community Support: ROS has a large and active community of developers and researchers, providing access to a wealth of resources, libraries, and ready-made solutions for robotics applications.

Considerations for Transitioning to ROS:

Learning Curve: Transitioning to ROS may require learning new concepts and tools, which can initially be challenging but ultimately rewarding in terms of the capabilities it offers.
Complexity: While ROS offers powerful features, it can be more complex than using Arduino and MIT App Inventor. Adequate training and resources may be needed to leverage ROS effectively.
Efficiency and Performance:
Utilizing ROS can improve the efficiency and performance of mobile robot manipulators by enabling more advanced control strategies, sensor fusion, and seamless integration with other robotic systems or components.

Thank You