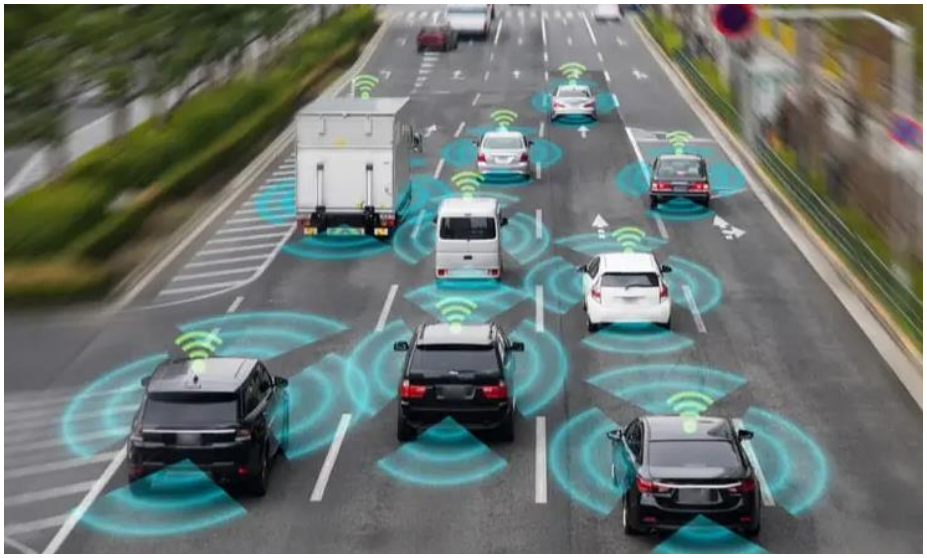
Ministry of Higher Education
Pyramids Higher Institute (PHI) for Engineering and Technology, 6th of October
Electronics and Communications Engineering Department

وزارة التعليم العالي
معهـــد الأهرامـــات العـــالي للهندســـة والتكنولوجيا بالسادس من أكتوبر
قسم هندسة الالكترونيات والاتصالات



Robot Modeling and Simulation

Report 1

Level 1 Autonomous Car - Modeling Implementation and Control

Group Name:

Blue Team

Supervisors:

Dr. Mohamed Hamdy El-Saify
Eng. Ahmed Abdel-Gawad

Objective:

To model, implement, and control a level 1 autonomous car based on range measurements taken from sensors positioned around the vehicle.

Introduction:

Self-driving vehicles, or autonomous driving, refer to cars' ability to navigate and operate without human intervention. This technology integrates sensors, algorithms, and actuators to perceive the environment, make decisions, and control the vehicle's movements. The importance of self-driving vehicles lies in their potential to revolutionize transportation by enhancing safety, efficiency, and accessibility. They can reduce human errors, a major cause of accidents, and improve traffic flow through optimized routes and coordinated vehicle movements. Moreover, self-driving vehicles offer mobility solutions for individuals unable to drive due to age or disability, contributing to greater independence and accessibility in transportation.

Problem statement and modeling:

1.1 Representing robot position

Throughout this analysis we model the robot as a rigid body on wheels, operating on a horizontal plane. The total dimensionality of this robot chassis on the plane is three, two for position in the plane and one for orientation along the vertical axis, which is orthogonal to the plane. Of course, there are additional degrees of freedom and flexibility due to the wheel axles, wheel steering joints, and wheel castor joints. However, by robot chassis we refer only to the rigid body of the robot, ignoring the joints and degrees of freedom internal to the robot and its wheels.

In order to specify the position of the robot on the plane, we establish a relationship between the global reference frame of the plane and the local reference frame of the robot, as in Fig. 1.1. The axes XI and YI define an arbitrary inertial basis on the plane as the global reference frame from some origin O: {XI , YI}. To specify the position of the robot, choose a point P on the robot chassis as its position reference point. The basis {XR , YR} defines two axes relative to P on the robot chassis and is thus the robot’s local reference frame. The position of P in the global reference frame is specified by coordinates x and y, and the angular difference between the global and local reference frames is given by θ. We can describe the pose of the

robot as a vector with these three elements. Note the use of the subscript I to clarify the basis of this pose as the global reference frame:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{1.1}$$

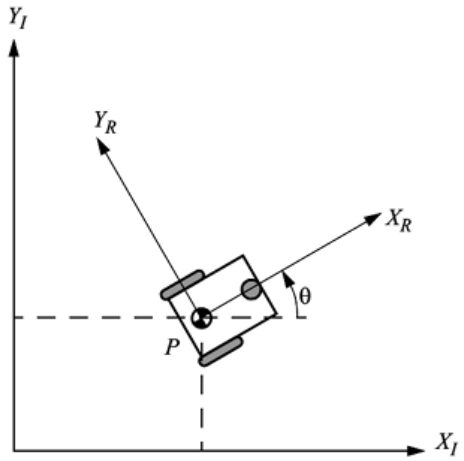


Fig. 1.1 The global reference frame and the robot local reference frame.

To describe robot motion in terms of component motions, it will be necessary to map motion along the axes of the global reference frame to motion along the axes of the robot's local reference frame. Of course, the mapping is a function of the current pose of the robot. This mapping is accomplished using the **orthogonal rotation matrix:**

$$\boldsymbol{R(\theta)} = \begin{bmatrix} \boldsymbol{cos(\theta)} & \boldsymbol{sin(\theta)} & \mathbf{0} \\ \boldsymbol{-sin(\theta)} & \boldsymbol{cos(\theta)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{1.2}$$

This matrix can be used to map motion in the global reference frame to motion in terms of the local reference frame {XR, YR}. This operation is denoted by $(\theta)\,\dot{\xi}_I$ because the computation of this operation depends on the value of:

$$\dot{\boldsymbol{\xi}}_{\boldsymbol{R}} = \boldsymbol{R(\theta)}\dot{\boldsymbol{\xi}}_{\boldsymbol{I}} \tag{1.3}$$

For example, consider the robot in Fig. 1.2. For this robot, because $\theta = \pi / 2$ we can easily compute the instantaneous rotation matrix R:

$$R\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.4}$$

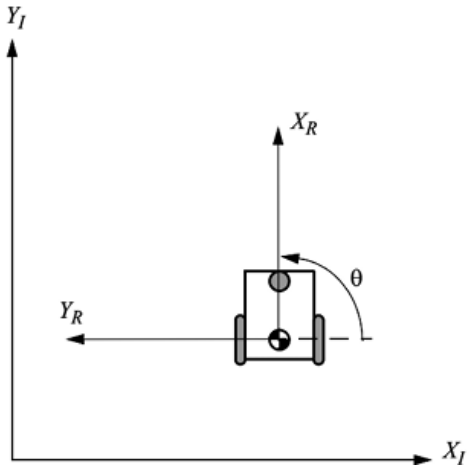


Fig. 1.2 The mobile robot aligned with a global axis.

Given some velocity (x,y,θ) in the global reference frame we can compute the components of motion along this robot's local axes XR and YR. In this case, due to the specific angle of the robot, motion along XR is equal to y, and motion along YR is $-\dot{x}$:

$$\dot{\xi}_R = R\left(\frac{\pi}{2}\right)\dot{\xi}_I = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ -\dot{x} \\ \dot{\theta} \end{bmatrix} \tag{1.5}$$

1.2 Forward kinematic models

In the simplest cases, the mapping described by equation (5.3) is sufficient to generate a formula that captures the forward kinematics of the mobile robot: how does the robot move, given its geometry and the speeds of its wheels? More formally, consider the example shown in Fig. 1.3.

This differential drive robot has two wheels, each with diameter r. Given a point P centered between the two drive wheels, each wheel is a distance l from P. Given r, l, θ, and the spinning speed of each wheel, $\dot{\varphi}_1$ and $\dot{\varphi}_2$, a forward kinematic model would predict the robot's overall speed in the global reference frame:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r_1, \theta, \dot{\varphi}_1, \dot{\varphi}_2) \tag{1.6}$$

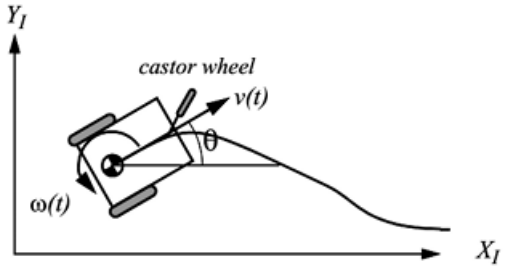


Fig. 1.3 A differential-drive robot in its global reference frame.

If wheel 1 spins alone, the robot pivots around wheel 2. The rotation velocity at can be computed because the wheel is instantaneously moving along the arc of a circle of radius 2l:

$$\omega_1 = \frac{r\dot{\varphi}_1}{2l} \tag{1.7}$$

The same calculation applies to the left wheel, with the exception that forward spin results in clockwise rotation at point P:

$$\omega_2 = \frac{-r\dot{\varphi}_2}{2l} \tag{1.8}$$

Combining these individual formulas yields a kinematic model for the differential-drive example robot:

$$\dot{\xi}_I = R(\theta)^{-1} \begin{bmatrix} \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2} \\ 0 \\ \frac{r\dot{\varphi}_1}{2l} + \frac{-r\dot{\varphi}_2}{2l} \end{bmatrix} \tag{1.9}$$

We can now use this kinematic model in an example. However, we must first compute $R(\theta)^{-1}$. In general, calculating the inverse of a matrix may be challenging. In this case, however, it is easy because it is simply a transform from $\dot{\xi}_R$ to $\dot{\xi}_I$ rather than vice versa:

$$R(\theta)^{-1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.10}$$

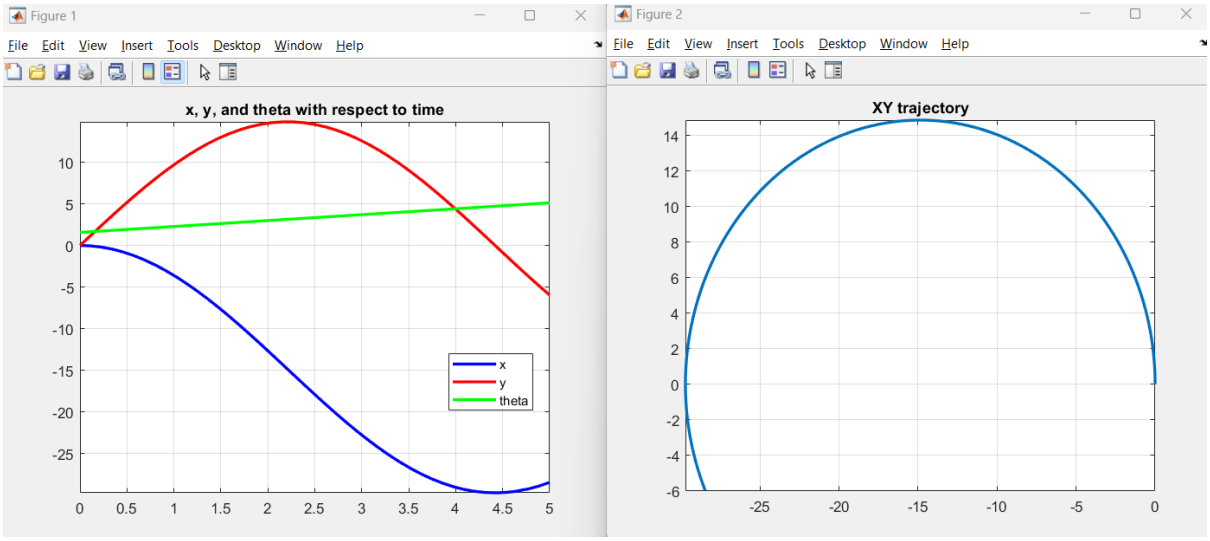
MATLAB CODE

```
% Clearing workspace, closing figures, and clearing command window
close all
clear
clc

% Parameters of the vehicle
r = 3.25; % wheel radius
l = 8;    % distance between wheel and center

% Wheel velocities
fhiRDot = 1.5; % right wheel velocity
fhiLDot = 5;    % left wheel velocity

% Initial position and orientation of the vehicle
x0 = 0;
y0 = 0;
theta0 = pi/2;

% Time span for simulation
tspan = [0 5];

% Calculating translational and rotational velocities
xrdot = (r/2) * (fhiRDot + fhiLDot);
yrdot = 0;
omegadot = (r/(2 * l)) * (fhiRDot - fhiLDot);

% Symbolic representation of position and orientation
syms x(t) y(t) theta(t)

% Defining kinematic differential equations
ode1 = diff(x) == cos(theta) * xrdot;
ode2 = diff(y) == sin(theta) * xrdot;
ode3 = diff(theta) == omegadot;
odes = [ode1; ode2; ode3];

% Initial conditions
cond1 = x(0) == x0;
cond2 = y(0) == y0;
cond3 = theta(0) == theta0;
conds = [cond1; cond2; cond3];
```

```
% Solving ODEs
S = dsolve(odes,conds);

% Displaying and plotting results
disp('x(t)=')
disp(S.x)
disp('y(t)=')
disp(S.y)
disp('theta(t)=')
disp(S.theta)

% Plotting x, y, and theta against time
fplot(S.x, tspan, 'b', 'linewidth', 2),  hold on
fplot(S.y, tspan, 'r', 'linewidth', 2)
fplot(S.theta, tspan, 'g', 'linewidth', 2),  grid on
legend('x', 'y', 'theta', 'Location', 'best')
title('x, y, and theta with respect to time')

% Plotting the XY trajectory
figure
fplot(S.x, S.y, tspan, 'linewidth', 2), grid on
title('XY trajectory')
```

CASE 1

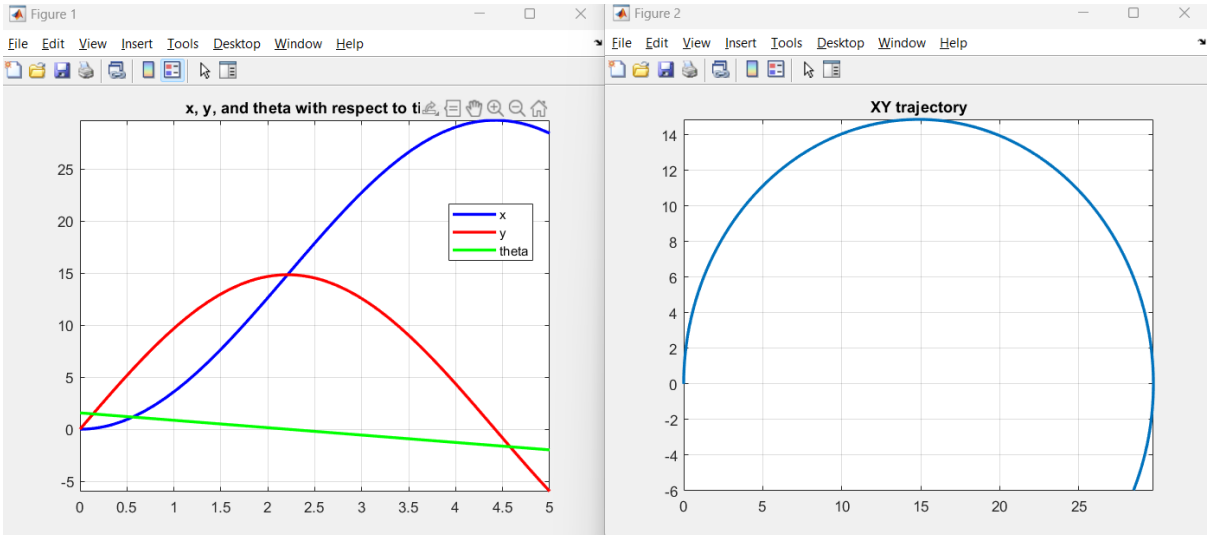
```
% Wheel velocities (Right wheel velocity > Left wheel velocity)
fhiRDot = 5; % right wheel velocity
fhiLDot = 1.5;    % left wheel velocity
```



$$x(t) = (104*\sin((91*t)/128 + pi/2))/7 - 104/7$$
$$y(t) = -(104*\cos((91*t)/128 + pi/2))/7$$
$$\text{theta}(t) = (91*t)/128 + pi/2$$

CASE 2

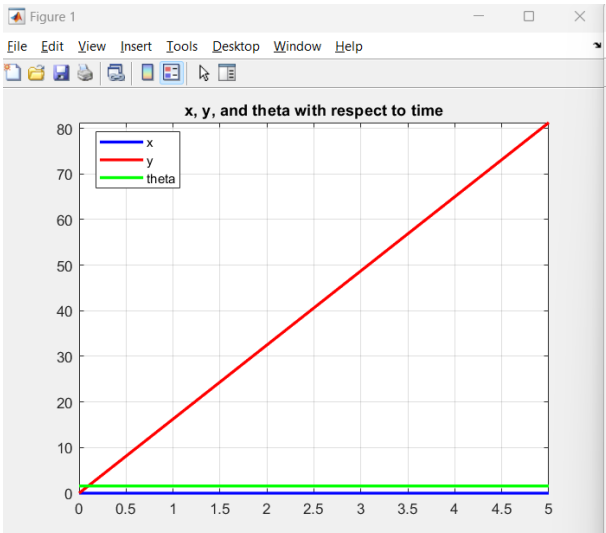
```
% Wheel velocities (Left wheel velocity > Right wheel velocity)
fhiRDot = 1.5; % right wheel velocity
fhiLDot = 5;    % left wheel velocity
```

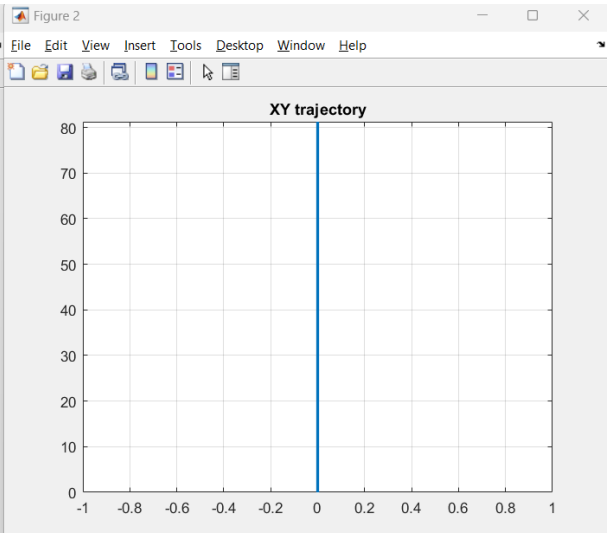


x(t) = (104*sin((91*t)/128 - pi/2))/7 + 104/7
y(t) = (104*cos((91*t)/128 - pi/2))/7
theta(t) = pi/2 - (91*t)/128

CASE 3

```
% Wheel velocities (Right wheel velocity = Left wheel velocity)
fhiRDot = 5; % right wheel velocity
fhiLDot = 5; % right wheel velocity
```





x(t) = 0
y(t) = (65*t)/4
theta(t) = pi/2

Control law:

Utilizing PID in autonomous driving systems can be beneficial for achieving precise control of the vehicle's movements. However, before tuning PID parameters (Proportional, Integral, Derivative), it's important to carefully choose the error term. Here are some points that can help in selecting the error term in a way that avoids oscillation between positive and negative values:

1. Sensor Validation: Ensure the accuracy of sensors used in the system, such as distance sensors. It's essential to isolate errors that might affect the actual error estimation.
2. Real-World Testing: Conduct practical tests of the system in a controlled environment, measure the actual error, and observe the system's response to this error.
3. Performance Monitoring: Utilize performance monitoring tools like data plots and time-domain curves to understand the impact of PID adjustments on system performance.

Calculate error

The difference (error) between the readings of the right ultrasonic sensor (UR) and the left ultrasonic sensor (UL)

1. **If UL is greater than UR (UL > UR):**

- Calculate the difference between UL and UR (difference = UL - UR). This difference represents how much closer the obstacle is to the left side compared to the right side.

2. **If UL is less than UR (UL < UR):**

- Calculate the difference between UR and UL (difference = UR - UL). This difference represents how much closer the obstacle is to the right side compared to the left side.

3. **If UL is equal to UR (UL == UR):**

- Set the difference to 0 (difference = 0), indicating that both sides are equidistant from an obstacle.

```
if (UL > UR) {
  dir = 'L';
  difference = UL - UR;
}
else if (UL < UR) {
  dir = 'R';
  difference = UR - UL;
}
```

```
else if (UR == UL) {
  dir = 'C';
  difference = 0;
}
```

Calculate Kp, Ki, Kd

The proportional (P), integral (I), and derivative (D) constants (Kp, Ki, and Kd) are crucial for the controller's performance.

1. **Proportional (P) term (Kp):**
   - We start by setting Kp to a small value, often around 0.1
   - Gradually increase Kp until the robot car starts responding to obstacles.
   - If the car reacts too slowly, increase Kp; if it's too sensitive, decrease Kp.
2. **Integral (I) term (Ki):**
   - We set Ki to zero initially and observe the car's response.
   - If there's a steady-state error, gradually increase Ki until the error diminishes.
   - If we set Ki too high as it may lead to overshooting or instability.
3. **Derivative (D) term (Kd):**
   - Initially, we set Kd to zero.
   - Increase Kd to reduce overshooting and oscillations caused by sudden changes in error.
   - If we set Kd too high, it can introduce noise amplification or even destabilize the system.

Hardware and software:

1. **Arduino UNO** is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button shown in Fig 2.1.

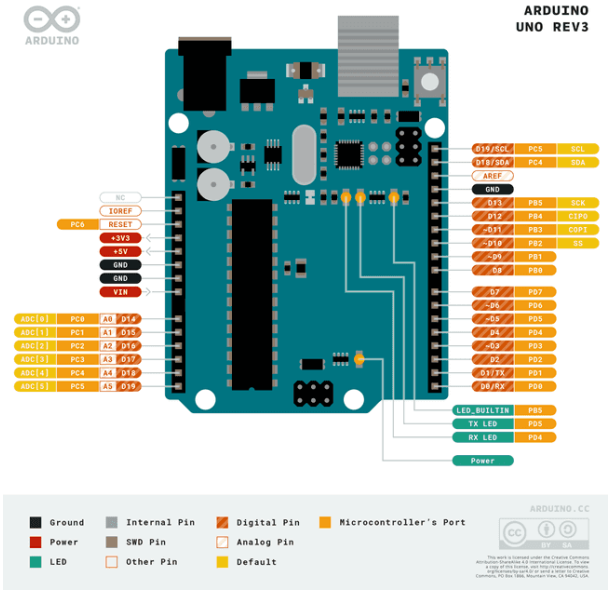


Fig 2.1 Arduino UNO pinout

2. **Ultrasonic sensors** measure distance using ultrasonic waves. The sensor head emits an ultrasonic wave and receives the wave reflected back from the target. Ultrasonic Sensors determine the distance to the target by measuring the time between the emission and reception shown in Fig 2.2.

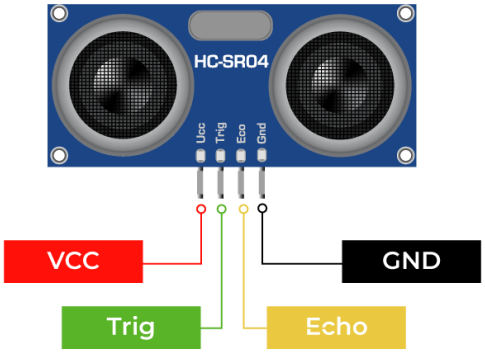


Fig 2.2 Ultrasonic sensors

3. **The L298 motor driver** is a popular dual H-bridge motor driver integrated circuit (IC) Shown in Fig 2.3 commonly used in robotics and other projects requiring motor control:
   - Dual H-Bridge design.
   - Maximum Voltage: up to 46V
   - Current Handling: The L298 can handle peak currents of up to 2A per channel and 4A if a heat sink is used.
   - Enable Pins: Each H-bridge has an enable pin that allows for PWM (pulse-width modulation) control of motor speed.

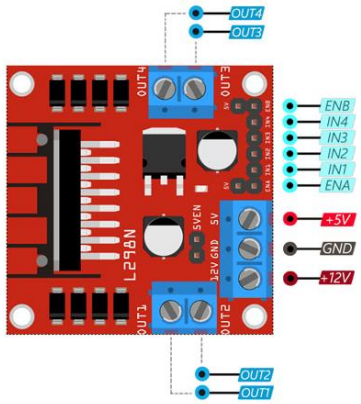


Fig 2.3 The L298 motor driver

4. A **DC geared motor with wheel** in autonomous cars is an electrical device that converts electrical energy into mechanical motion, crucial for functions like propulsion and steering. It operates on direct current (DC) power and is used in various systems within autonomous vehicles, such as propulsion motors for driving wheels or steering motors for directional control. These motors enable the car to move, accelerate, decelerate, and navigate autonomously, playing a key role in the vehicle's overall functionality and movement capabilities shown in Fig 2.4

Fig 2.4 DC geared motor with wheel

5. In autonomous cars, the **rigid body** serves as a stable platform for mounting sensors such as ultrasonic and electronic components, ensuring accurate data collection. This setup minimizes vibrations and noise, which are crucial for sensor precision and system reliability. The design also enhances collision resistance and impact absorption, safeguarding internal components. Treating the car as a rigid body simplifies the mathematical analysis for motion dynamics evaluations, as shown in Figure 2.5.

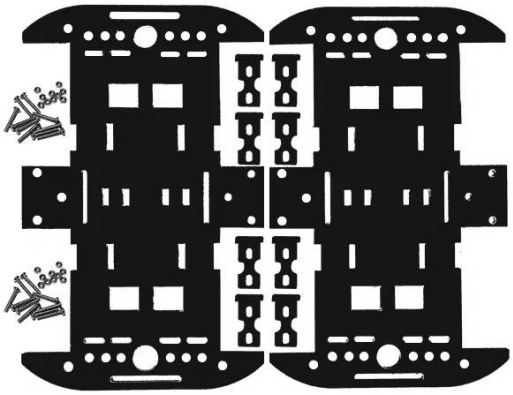


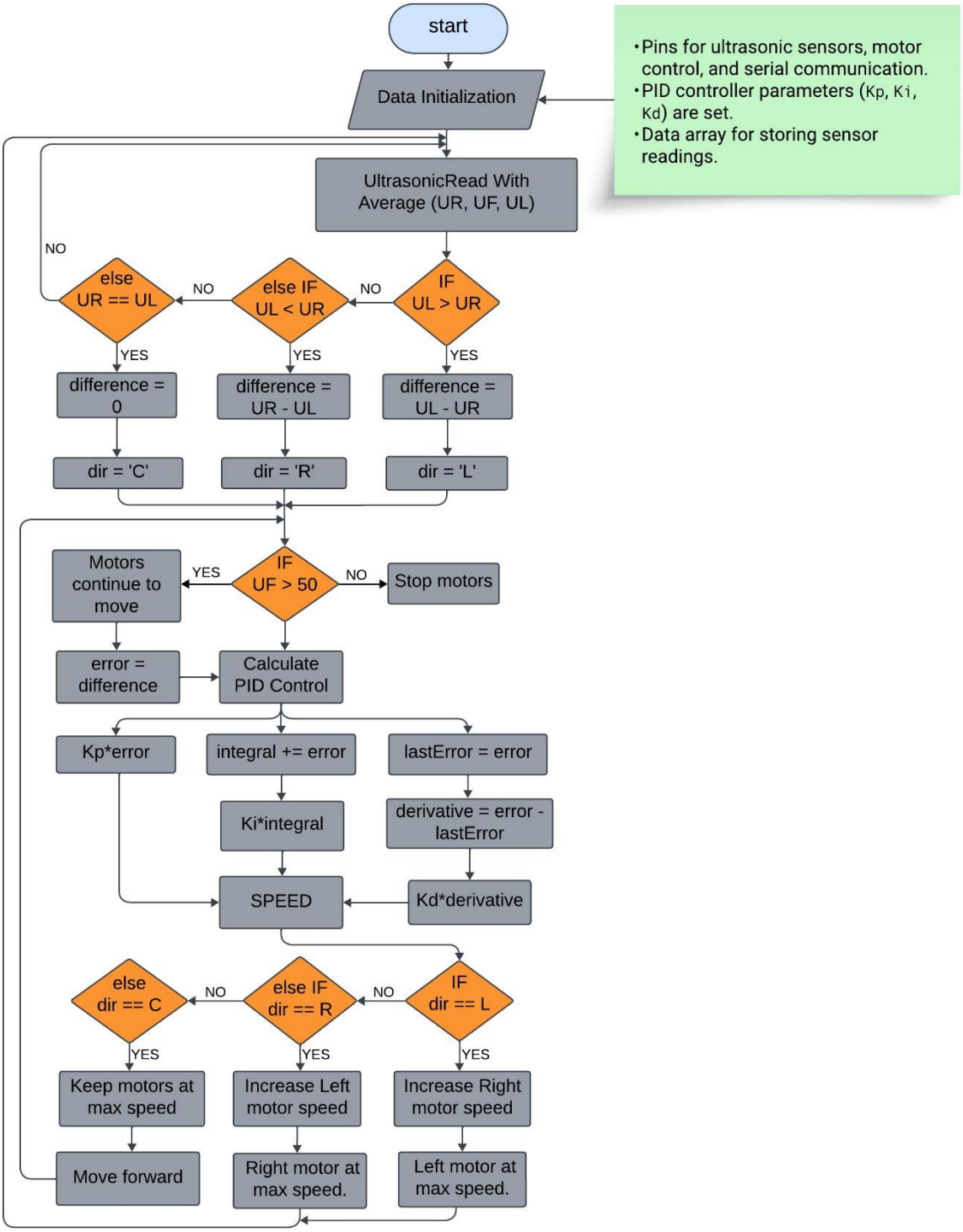
Fig 2.5 The Rigid body

6. The **Li-Ion 18650 battery** is a compact and efficient power source widely utilized across various industries. Featuring a typical capacity of 2000 mAh and a nominal voltage of 3.7V, it offers reliable performance for a multitude of electronic devices. Its flat top positive cap and rechargeable nature enhance usability and longevity.

   With dimensions of 65mm in length and 18mm in diameter, the Li-Ion 18650 battery shown in Fig 2.6 is designed to fit seamlessly into a range of applications without compromising on power output. Furthermore, its safety features include a discharge end voltage of 2.5V and a charging maximum voltage of 4.20V, ensuring optimal performance and longevity.

Fig 2.6 Li-Ion 18650 battery

FlowChart


start
Data Initialization
•Pins for ultrasonic sensors, motor control, and serial communication.
•PID controller parameters (Kp, Ki, Kd) are set.
•Data array for storing sensor readings.
UltrasonicRead With Average (UR, UF, UL)
IF UL > UR
else IF UL < UR
else UR == UL
NO
YES
difference = UL - UR
difference = UR - UL
difference = 0
dir = 'L'
dir = 'R'
dir = 'C'
IF UF > 50
Stop motors
Motors continue to move
Calculate PID Control
error = difference
lastError = error
integral += error
Kp*error
derivative = error - lastError
Ki*integral
Kd*derivative
SPEED
IF dir == L
else IF dir == R
else dir == C
Increase Right motor speed
Increase Left motor speed
Keep motors at max speed
Left motor at max speed.
Right motor at max speed.
Move forward

Results and analysis:

The car was tested on a curved track for accuracy and speed, avoiding collisions but lacking desired accuracy. To improve, a PID control system was installed, and Kd, Ki, and Kp values were adjusted iteratively. Through multiple tests on curved tracks, the car achieved both the desired speed and obstacle avoidance, showcasing the effectiveness of the PID control system in enhancing performance and accuracy in challenging driving scenarios.


Figure Turn 1

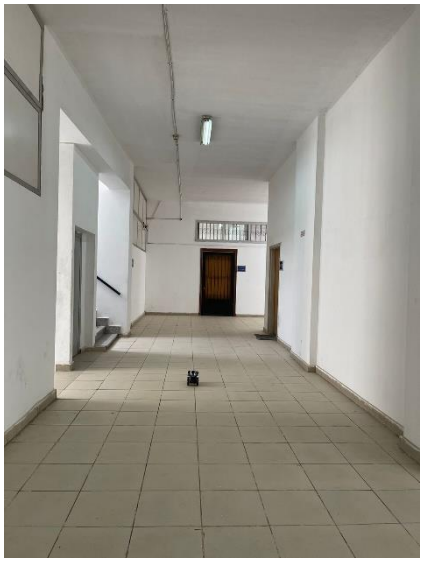


Figure Track 1


Figure Turn 2


Figure Track 2


Figure Turn 3


Figure Track 3


Figure Turn 4

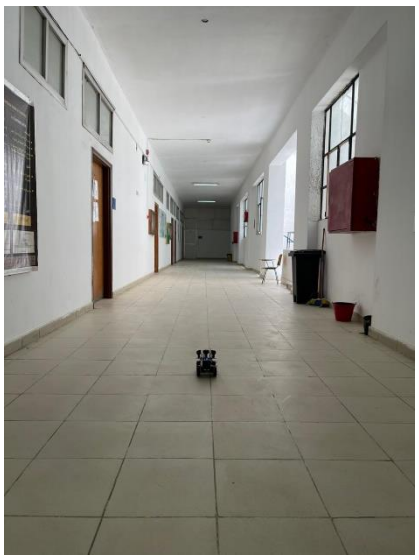


Figure Track 4

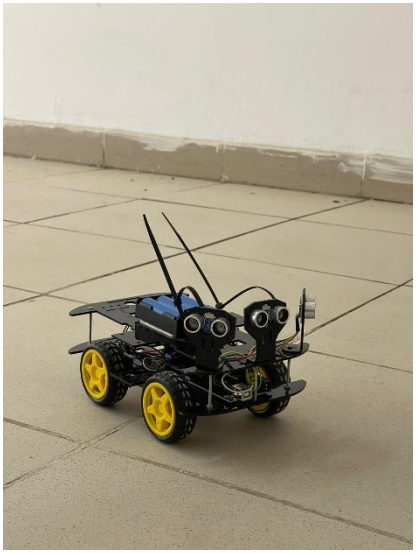


Figure Robot car

Conclusion:

In conclusion, the development and implementation of a self-driving robotic car using MATLAB modeling and simulation provides a powerful platform for designing and testing self-driving car control algorithms and behavior. Through mathematical models and simulations, engineers can replicate and improve a vehicle's performance under different conditions, enhancing its reliability and safety before deploying it in the real world. Integrating PID control into the control system of a self-driving car plays a pivotal role in achieving precise and stable movements. The integration of ultrasonic sensors adds an important dimension to the environmental awareness of the self-driving car. It helped us to test the car on a road full of obstacles, and the car succeeded in avoiding obstacles with reasonable results.

Appendix:

GitHub Code

https://github.com/MahmoudElbhrawy/Level-1-Autonomous-Car

Students Names	Codes
Abobakr Mostafa Fathy	20200276
Ahmed Sayed Mahmoud	20200018
Eslam Ahmed Ibrahim	20200107
Esraa Yasser Mohamed	20200169
Asmaa Mohamed Abdelhamed	20200021
Basent Abdelhamed Mohamed	20200026
Madlen Nady Samir	20200020
Mohamed Fadl Sholkamy	20200144
Mohamed Hany Hassan	20200129
Mahmoud Mohamed Elbhrawy	20200206
Moaaz Mohamed Mohamed	20190257
Youssef Ehab Mohamed	20190100
Youssef Said Ibrahim	20190159