



Addis Ababa Institute of Technology

School of Electrical and Computer Engineering
Autonomous Vacuum Cleaner Robot

By: Biruk Berhanu
Advisors' Name: Mr. Kinde Mekuria
Date of submission: Dec 14, 2020

Contents

Abstract	1
Acknowledgment	2
Acronym.....	3
Chapter 1 Overview	4
1.1 Introduction and Background.....	4
1.2 Literature Review	4
1.2.1 Mechanical Design	5
1.2.1 Control Theory	6
1.2.2 Path Planning.....	7
1.3 Problem Statement	8
1.4 Objective	8
1.5 Thesis Organization.....	9
Chapter 2 Mathematical Modeling	10
2.1 Introduction	10
2.2 Differential Drive Kinematic	10
2.3 Forward Kinematics	11
2.4 Reverse Kinematics.....	12
2.5 Odometry.....	13
Chapter 3 Mechanical Structure and Hardware Design.....	18
3.1 Wheels and Chassis	18
3.2 Cleaning Structure.....	20
3.3 Perception.....	21
3.4 Control sensors and microcontroller	23
3.5 Power source	25
3.6 Physical Dimension.....	27
3.7 Feasibility analysis	28
Chapter 4 Control System Design.....	29
4.1 Sampling time	29
4.2 Design Architecture.....	29

4.2.1 The Unicycle Model	30
4.2.2 High level model	31
4.3 Go-to-Goal behavior	32
4.4 Obstacle avoidance.....	33
Chapter 5 Testing, Simulation Result, and Conclusion	36
5.1 Result.....	36
5.1.1 Implementation of Control	36
5.1.2 Hardware	39
5.2 Conclusion.....	42
Chapter 6 References	44
Chapter 7 Appendix A	45
7.1 Odometry implementation in Arduino	45
7.2 Conversion between unicycle and differential drive model.....	46
7.3 Go to Goal implementation	46

List of Tables

Table 3.1 Maximum current usage of all the component while power on.....	25
Table 3.2 Physical dimension of the different parts of the robot	27

List of Figures

Figure 1:1 Behavior based robotic.....	6
Figure 2:1 Differential Drive kinematics (from Dudek and Jenkin, <i>Computational Principles of Mobile Robotics</i> ...)	10
• Figure 2:2 a Optical wheel encoder. b Encoding disk (Mordechai Ben-Ari Robotic Motion and Odometry, Fig5.7)	13
Figure 2:3 Odometry geometry. Over a small time period, the robot's motion can be approximated by an arc. The odometry problem is to solve for (x_0, y_0, θ_0) given (x, y, θ) and $dbaseline$. In the figure, the robot is moving counter-clockwise. A Primer on Odometry and Motion control, Edwin Olson.....	14
Figure 3:1 Solid work design exploded view of reclean.....	18
Figure 3:2 Solid work design 3D view of RoboClean	19
Figure 3:3 Solid work design 3D view captured from bottom of RoboClean	19
Figure 3:4 Solid work design 3D exploded view.....	19
Figure 3:5 Cleaning section of RoboClean.....	20
Figure 3:6 Dust bin with a removable plate to remove the dust	20
Figure 3:7 Bottom view of the dust bin while the removable plate moving	20
Figure 3:8 Distance measurement angle using servo and ultrasonic	21
Figure 3:9 NodeMCU operating as an access point(Source:LastMinuteEngineering.com)	24
Figure 3:10 F249 Infrared speed sensor used for measuring speed and position.....	25
Figure 3:11 Speed encoder disk.....	25
Figure 3:12 Power Bank	26
Figure 3:13 Inside the USB cable wire.....	27
Figure 4:1 Example of a unicycle drive.....	30
Figure 4:2 The Simplified Unicycle Robot Model	30
Figure 4:3 A robot and the control vector.....	31
Figure 4:4 function used to regulate the value of u using the value of e	32
Figure 4:5 Robot's motion reaction near to obstacles.....	33
Figure 4:6 Sensor's coordinates are in the robot's frame	33
Figure 4:7 Ultrasonic distance defined in the sensor's frame	34
Figure 4:8 Robot's coordinates are in the world frame	35

Figure 5:1 Position of Robot while moving from (0,0) to (150,150).....	37
Figure 5:2 performance of the robot angle tracking while moving from position (0,0) to (150,150)	37
Figure 5:3 performance of the robot moving on a straight line for PID of $K_p = 2$, $K_i = 0.05$, $K_d = 0$	38
Figure 5:4 performance of the robot moving on a straight line for PID of $K_p = 2.5$, $K_i = 0.5$, $K_d = 0.1$	38
Figure 5:5 performance of the robot moving on a straight line for PID of $K_p = 20$, $K_i = 10.$, $K_d = 5$	39
Figure 5:6 Phase one implementation of RoboClean	39
Figure 5:7 Phase two implementation of RoboClean	40
Figure 5:8 Phase three implementation of RoboClean	41
Figure 5:9 Last phase implementation of RoboClean.....	41
Figure 5:10 Torque for a given speed of a dc motors(courtesy: https://blog.orientalmotor.com/eliminate-motor-speed-fluctuations-due-to-input-voltage-or-load-variance)	43

Abstract

We are in the age of the new era, which is the age of the digital world. Almost everything has become autonomous and accompanied by AI. I strongly argue not to do house chores manually since we have already passed the manual age. There is some progress in building an Autonomous vacuum cleaner robot to clean our houses merely without any help from people. But current Vacuum cleaner robots are somewhat expensive and inefficient. “It is, therefore, a sine qua non to improve the technology of vacuum cleaning to reduce these deficiencies” [1].

In this thesis I present the development of compact, efficient, and cheap vacuum cleaner robot. The robot will be disk-shaped, equipped with vacuuming and cleaning technology and controlled by Arduino Mega microcontroller. It sucks dirt via a retractable dustbin on top of which a cooling fan is mounted, and one sweepers driven by DC motor. The robot navigates using two DC geared motor mounted at the center axis of the robot. There will also be a third rear caster wheel which is used for balancing and ease of control. The robot will be modeled using differential drive model. The perception of the robot is accompanied by one ultrasonic module mounted on a servo simulating a radar and two optical encoders connected to the two-axis wheel to count the number revolution made by the wheel.

Acknowledgment

I would wish to express my special thanks of gratitude to my advisor Mr. Kinde Makaria who gave me the opportunity to do this project on the topic Autonomous Vacuum Cleaner Robot, which also helped me in understanding a lot of ideas and approach. Secondly, I would also like to thank my friends from mechanical engineering department who helped me on designing the structural aspect of the robot.

Acronym

RoboClean ---- Robot Cleaner

DC --- Direct Current

pose --- position and orientation

Chapter 1 Overview

1.1 Introduction and Background

A very notable household chore is floor cleaning which is usually considered as unpleasant, difficult, awkward and boring. In most cases, cleaners are hired to do the task rather than the household residents do it. The discomfort posed by this recurrent chore necessitated development of a vacuum cleaner that could assist human with such a task. A vacuum cleaner is an electromechanical appliance commonly used for cleaning floors, furniture, rugs and carpets by suction. An electric motor inside the appliance turns a fan which creates a partial vacuum and causes outside air to rush into the evacuated space. This forces any dirt or dust near the nozzle into a bag inside the machine or attached to the surface(Asafa et al. 2018).

The robotic vacuum is mainly built from an easily available material and limited number electronic devices. The robotic vacuum uses a rotating sweeper underneath the unit to vacuum a floor as it passes over it. Two DC motors, aligned across the center axis of the robot, are used to accurately drive the robotic vacuum around a room. Because the body of the robot is circular and the DC are placed along the center axis, the robot can spin in place in any direction. One caster wheel is located at the rear of the robot to keep it balanced.

The robot is programmed to sense the direction of a collision with an obstacle using an ultrasonic sensor mounted on a servo motor. The robotic vacuums movement is based upon two navigations strategies. The first one is a random walk around a room, which theoretically will cover the entire area of a room given enough time. The robot uses the ultrasonic sensor to avoid obstacle. It moves in a strait direction until it faces an obstacle then It will make a random angle turn and drive away from the obstacle. The second approach to the navigation problem is to drive the robot in a predefined path. The robot will track of the position and orientation as it maneuvers. In addition, it will also avoid an obstacle and get back to the predefined paths.

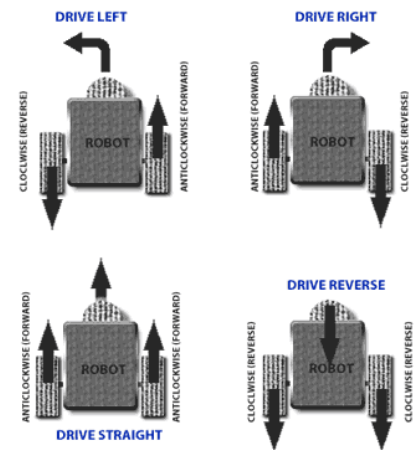
1.2 Literature Review

During my research on previously made vacuum cleaner robots, I found out that there has not been made a significant investigation on this area of subject specifically. I think this is due to the behavior of the subject field. Autonomous vacuum cleaner is a subject composed of control theory, path planning, Artificial intelligence, electronic circuit design and mechanical design. There are significant amounts of research papers on the individual aspect of the above subjects. The goal of this literature review is to compare the different approach studied by each field.

1.2.1 Mechanical Design

There are some different mechanisms to implement the platform(chassis) of the robot. For example, Differential Drive, Skid steering, Tricycle Drive, Ackermann steering.

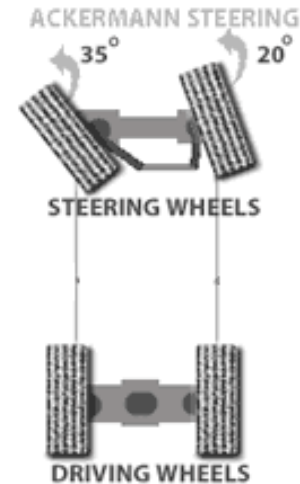
- **Differential Drive:** This is the most common control mechanism for robot builders, especially for beginners. The concept is simple; Velocity difference between two motors drive the robot in any required path and direction. Hence the name “**Differential**” drive. Differential wheeled robot can have two independently driven wheels fixed on a common horizontal axis or three wheels where two independently driven wheels and a roller ball or a castor attached to maintain equilibrium [2].



- **Tricycle Drive :** Tricycle robot is designed with a front steering wheel controlled by a motor. The two rear wheels are attached to a common axle driven by a single motor with two degrees of freedom (either forwards or reverse). The disadvantage is that they cannot spin like a differential drive robot and does not have a 90° turn due to their limited radius of curvature. Few robots have both steer and drive controlled by the front wheel and the rear wheels act as supporting wheels to maintain equilibrium. Due to the design constraints and its drawbacks, this design is less appreciated by the robot building community [2].



- Ackermann steering:** One of the most common configurations found in cars is Ackerman steering which mechanically coordinates the angle of two front wheels which are fixed on a common axle used for steering and two rear wheels fixed on another axle for driving. The advantage in this design is increased control, better stability and maneuverability on road, less slippage and less power consumption. This might resemble the tricycle approach where the front wheel is replaced with two wheels and an axle. But when two wheels are attached to a tricycle design (axle-articulated drive), then turning causes the robot to skid. To overcome that drawback Ackerman steering is designed in such a way that when there is a turn, the inner tire turns with a greater angle than the outer tire and avoids tire slippage. This approach can be generally used for fast outdoor robots which require excellent ground clearance and traction. Although there are disadvantages, the downside is additional parts required; no zero radii turn and increased complexity in design [2].



1.2.1 Control Theory

Mobile robots perform their intended task usually in a dynamic environment, this implies that the control for this kind of system is complex. One of the most chosen approach is to divide and conquer the different possible occurrence of environment and to design a control for each possibility. This approach is well studied and is called Behavior-Based-Robotics.

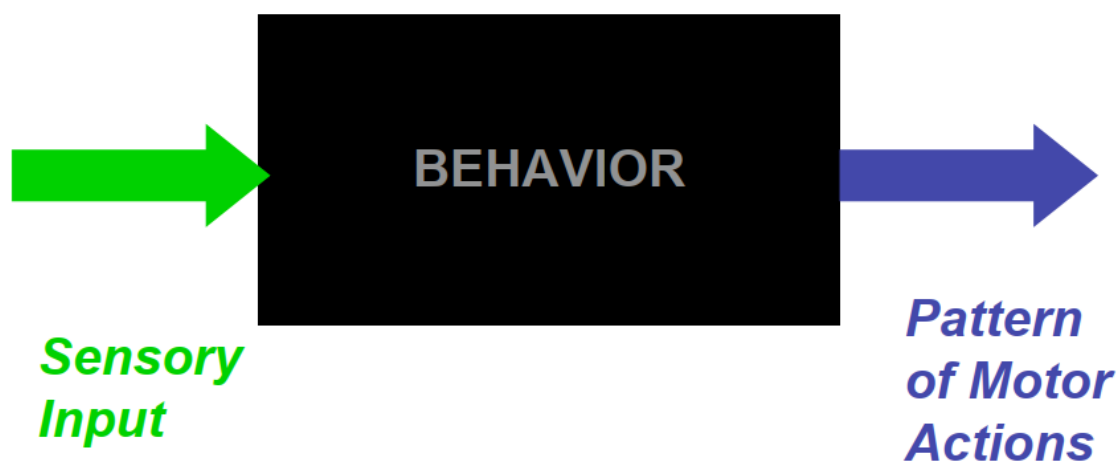


Figure 1:1 Behavior based robotic

Types of behaviors

- Reflexive : stimulus-response, often abbreviated S-R

“A reactive robotic system tightly couples perception to action without the use of intervening abstract representations or time history.” --Ron Arkin

- Reactive : learned or “muscle memory”
- Conscious : deliberately stringing together

1.2.2 Path Planning

Path-planning is an important task for mobile robots that lets robots find the shortest path between two points. The algorithm implementing the path planning should also choose an optimal path. Optimal paths are those that minimize the amount of turning, breaking or whatever a specific application requires. Map of the environment and the awareness of the robot’s location with respect to the map are the requirement for path planning. A*, D*, and RRT are the most commonly used path-planning algorithm.

Putting it simply, to solve the robot navigation problem, we need to find answers to the three following questions: Where am I? Where am I going? How do I get there? These three questions are answered by the three fundamental navigation functions localization, mapping, and motion planning, respectively.

- **Localization** : It helps the robot to determine its location in the environment. Numerous methods are used for localization such as cameras [3], GPS in outdoor environments [4], ultrasound sensors [5], laser rangefinder . The location can be specified as symbolic reference relative to a local environment (e.g., center of a room), expressed as topological coordinate (e.g., in Room 23) or expressed in absolute coordinate (e.g., latitude, longitude, altitude).
- **Mapping** : The robot requires a map of its environment in order to identify where he has been moving around so far. The map helps the robot to know the directions and locations. The map can be placed manually into the robot memory (i.e., graph representation, matrix representation) or can be gradually built while the robot discovers the new environment. Mapping is an overlooked topic in robotic navigation.
- **Motion planning or path planning** : To find a path for the mobile robot, the goal position must be known in advance by the robot, which requires an appropriate addressing scheme that the robot can follow. The addressing scheme serves to indicate to the robot where it will go starting from its starting position. For example, a robot may be requested to go to a certain room in an office environment with simply giving the room number as address. In other scenarios, addresses can be given in absolute or relative coordinates.

1.3 Problem Statement

Currently, in an urban area, most people spent their time doing their day-to-day jobs. In addition to their day-to-day jobs, they are also required to do their household chores. Naturally, household chores are boring and it would not be fair for a person to engage in such kind of task after a long working day. A vacuum cleaner robot will eliminate one of the tiresome chores, which is cleaning a house. Therefore, a lot of busy people would benefit from the autonomous vacuum cleaner.

Cleaning of these offices and buildings happen at dawn, which is a time when peoples start to crowd the place. During this time there is a smell of dust and detergent, which is very uncomfortable and unhealthy. However, most offices and buildings are not used at night. Therefore, if cleaning is made at the time when there are no people, for example starting from midnight, it would eliminate the above problem. The solution can be achieved using an autonomous vacuum cleaner robot.

Cleaning home has never been easy for older peoples and disables. They need a form of a robot that performs the task. Implementing the vacuum cleaner robot would help a lot with such kind of community.

Depending on the design target, robotic vacuum cleaners are appropriate for offices, hotels, hospitals, and homes. However, most cheap cleaners need a better cleaning pattern algorithm for efficient functioning while the smart ones are rather costly, and thus beyond the reach of most homes. These challenges were carefully considered while designing the robot vacuum cleaner described in this paper(Asafa et al. 2018).

As a final project, I made a decision to design and build a robot capable of vacuuming the floor of a room or area with no human interaction aside from just starting the unit. I realized the need for a cheap and convenient product that can be easily used to vacuum a room on its own, saving a person valuable time.

1.4 Objective

Generally making a robot requires some parameters for consideration and it can have many scopes on how deep the robot is intended to do the required task. On this project, it is planned to have the following feature:

- a. It will have two DC motor for its movement with the help of one caster wheel for balancing
- b. It will have one motor to create a vacuum at the bottom of the robot
- c. It will have one ultrasonic sensor mounted on a servo motor for obstacle detection and navigation
- d. It will have one DC motor for implementing the function of sweeper

- e. Two mode of operation:
 - 1. The first mode of operation is without any control from the external environment. This mode of operation is what makes the robot autonomous.
 - 2. The second mode of operation is by using a web browser on phone or desktop to monitor and navigate incase the robot got stuck at some place.

1.5 Thesis Organization

1. **Mathematical Modeling:** This chapter discuss about the mathematical model of the robot. The robot is modeled as a differential drive robot. This driving mechanism is used by a lot of mobile robots. It consists of two drive wheels mounted on a common axis, and each wheel can independently be driven either forward or backward.
2. **Mechanical Structure and Electrical Hardware Design:** This chapter discuss about the mechanical structure and Electrical hardware design of the robot. It consists of CAD design for the mechanical structure and schematic and circuit diagram for the Electrical hardware design.
3. **Control System Design:** This chapter focuses on the fundamental control of the robot. Due to performance and simplicity advantage, the control of the robot is divided into different behaviors. Behaviors are specific targeted tasks the robot performs. The robot control jumps from one behavior to another behavior based on the environment condition. The robot will have three behaviors, Go-To-Goal, Obstacle-Avoidance, and Wall-Following. The robot can navigate any kind of environment by switching from one behavior to another behavior.
4. **Testing and Simulation Result:** This will be the last chapter for presenting results on hardware design, software design and the robot control design.

Chapter 2 Mathematical Modeling

2.1 Introduction

The differential drive is a two-wheeled drive system with independent actuators for each wheel. The name refers to the fact that the motion vector of the robot is the sum of the independent wheel motions. The drive wheels are usually placed on each side of the robot and toward the front [6].

Even though differential drive robots' benefits, there are some points to be aware of before designing our system. They are very sensitive to slight change in velocity in each wheel, small error in the relative velocities between the wheels can affect the robot trajectory. They are also very sensitive to slippery floor, and finally, caster wheels for balancing is necessary.

2.2 Differential Drive Kinematic

(Dudek and Jenkin, *Computational Principles of Mobile Robotics*) Many mobile robots use a drive mechanism known as differential drive. It consists of 2 drive wheels mounted on a common axis, and each wheel can independently be driven either forward or backward. While we can vary the velocity of each wheel, for the robot to perform rolling motion, the robot must rotate about a point that lies along their common left and right wheel axis. The point that the robot rotates about is known as the ICC - Instantaneous Center of Curvature (see figure 1)

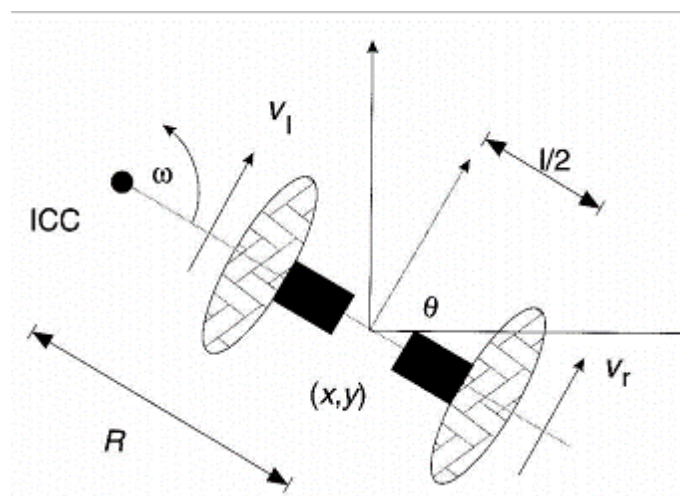


Figure 2:1 Differential Drive kinematics (from Dudek and Jenkin, *Computational Principles of Mobile Robotics*).

By varying the velocities of the two wheels, we can vary the trajectories that the robot takes. Because the rate of rotation ω about the ICC must be the same for both wheels, we can write the following equations:

$$\begin{aligned}\omega (R + l/2) &= V_r \\ \omega (R - l/2) &= V_l\end{aligned}$$

where l is the distance between the centers of the two wheels, V_r ; V_l are the right and left wheel velocities along the ground, and R is the signed distance from the ICC to the midpoint between the wheels. At any instance in time, we can solve for R and ω :

$$R = \frac{l}{2} \frac{V_l + V_r}{V_r - V_l}; \quad \omega = \frac{V_r - V_l}{l};$$

There are three interesting cases with these kinds of drives [7].

1. If $V_l = V_r$, then we have forward linear motion in a straight line. R becomes infinite, and there is effectively no rotation - ω is zero
2. If $V_l = -V_r$, then $R = 0$, and we have rotation about the midpoint of the wheel axis - we rotate in place.
3. If $V_l = 0$, then we have rotation about the left wheel. In this case $R = 2l$. Same is true if $V_r = 0$.

2.3 Forward Kinematics

[7]The forward kinematics equations for a robot (or other vehicle) with differential drive are used to solve the following problem: Standing in the pose (x, y, θ) at time t , determine the pose (x', y', θ') at time $t + \delta t$ given the control parameters V_l and V_r . The solution is typically used for automatic control of a robot such that it follows a wanted trajectory.

In General, we can describe the position of the robot capable of moving in a particular direction θ_t at a given velocity $V(t)$ as:

$$\begin{aligned}
x(t) &= \int_0^t V(t) \cos[\theta(t)] dt \\
y(t) &= \int_0^t V(t) \sin[\theta(t)] dt \\
\Theta(t) &= \int_0^t \omega(t) dt
\end{aligned}$$

For the special case of a differential drive, the above equations become:

$$\begin{aligned}
x(t) &= \frac{1}{2} \int_0^t [v_r(t) + v_l(t)] \cos[\theta(t)] dt \\
y(t) &= \frac{1}{2} \int_0^t [v_r(t) + v_l(t)] \sin[\theta(t)] dt \\
\Theta(t) &= \frac{1}{l} \int_0^t [v_r(t) - v_l(t)] dt
\end{aligned}$$

2.4 Reverse Kinematics

While the forward kinematics equations provide an updated pose given certain wheel speeds (or encoder counts), we can also formulate an inverse problem: Standing in pose (x, y, θ) at time t , determine control parameters V_l and V_r such that the pose at time $t + \delta t$ is (x', y', θ') .

This problem most often has no solution, in the sense that the robot cannot reach an arbitrary pose by simply setting appropriate values for wheels speeds V_l and V_r and let the motors run for a while. For some robots and vehicles, it is possible, and these vehicles are called Holonomic. However, most vehicles and robots are non-holonomic. For instance, a car is non-holonomic and this is why pocket parking is so hard.

For a non-holonomic robot, there are ways to increase the constrained mobility. If we allow a sequence of different (V_l, V_r) . There are normally infinitely many ways to move from one pose to another.

The following algorithm can be used to reach any target pose from any starting pose:

1. Rotate until the robot's orientation coincides with the line from the starting position to the target position: $V_r = -V_l = V_{rot}$
2. Drive straight until the robot's position coincides with the target position: $V_r = V_l = V_{ahead}$
3. Rotate until the robot's orientation coincides with the target orientation: $V_r = -V_l = V_{rot}$

Where, V_{rot} and V_{ahead} can be chosen arbitrarily.

We can also make the task as a closed loop system with a controller and feedback mechanism. In this method the system will go to the goal location while reducing the error between the present pose and goal pose. Both approaches are implemented and will be discussed in Chapter 4.

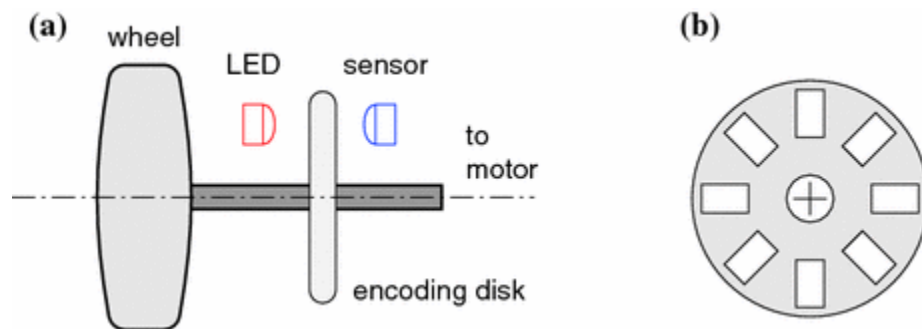
2.5 Odometry

Most robotics problems ultimately reduce to the question: where am I? How can you go somewhere else without some notion of where you are now? You could proceed naively, stumbling around, hoping that your goal will appear in range of your robot's sensors, but a better plan is to navigate. A basic method of navigation, used by virtually all robots, is odometry, using knowledge of your wheel's motion to estimate your vehicle's motion [8].

In practice there are significant number of ways to implement odometry for robots. We can categorize most of the method into two categories based on the sensor used

1. Using External Sensor : ways of getting information about the position and orientation from outside
 - a. Ultrasonic
 - b. Infrared
 - c. Camera
 - d. Laser scanner
 - e. GPS
2. Using Internal Sensor : ways of getting information about the position and orientation by calculating inside the system
 - a. Compass (for heading)
 - b. Accelerometers, Gyroscopes (for position)
 - c. Wheel encoder (both for heading and position)

The robot presented in this thesis will use optical Encoder and wheel encoder for odometry.



- Figure 2:2 **a** Optical wheel encoder. **b** Encoding disk (Mordechai Ben-Ari Robotic Motion and Odometry, Fig5.7)

Wheel Encoder

[8] Suppose our robot is at (x, y, θ) . Depending on wheel encoder sensor, we get measurements of how much the motors have rotated (in radians). Given the amount of rotation of the motor and the diameter of the wheel, we can compute the actual distance that the wheel has covered. Given the amount of rotation of the motor and the diameter of the wheel, we can compute the actual distance that the wheel has covered.

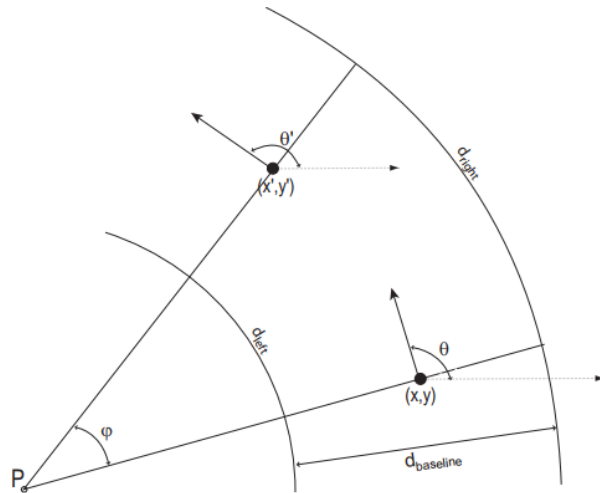


Figure 2:3 Odometry geometry. Over a small time period, the robot's motion can be approximated by an arc. The odometry problem is to solve for (x_0, y_0, θ_0) given (x, y, θ) and $d_{baseline}$. In the figure, the robot is moving counter-clockwise. A Primer on Odometry and Motion control, Edwin Olson

The initial state (x, y, θ) defines our starting point, with θ representing the vehicle's heading. After our vehicle has moved by d_{left} and d_{right} , we want to compute the new position, (x_0, y_0, θ_0) . The center of the robot (the spot immediately between the two wheels that defines the robot's location), travels along an arc as well. Remembering that arc length is equal to the radius times the inner angle, the length of this arc is:

$$d_{center} = \frac{d_{left} + d_{right}}{2}$$

Given basic geometry, we know that:

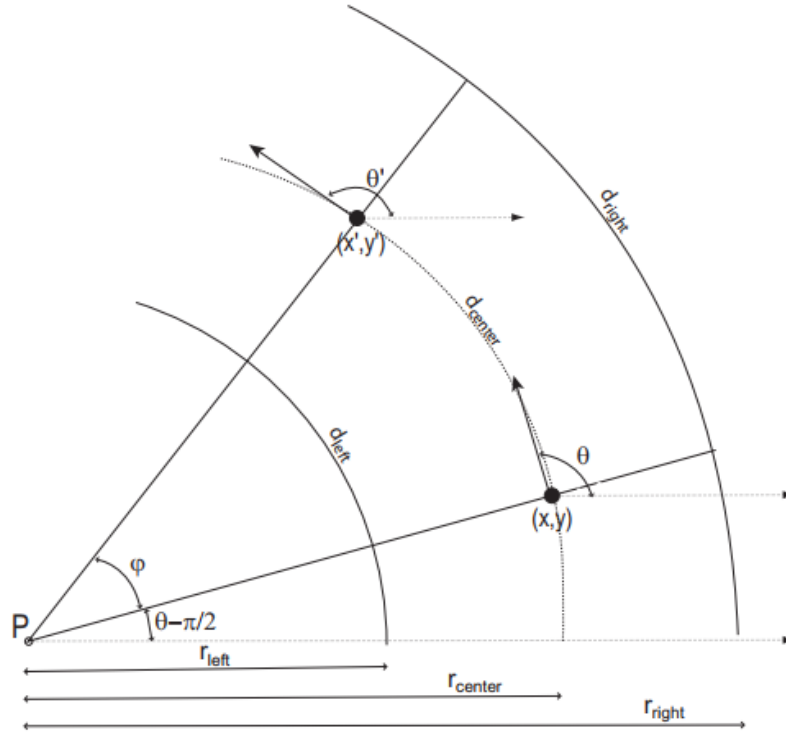
$$\begin{aligned}\phi r_{left} &= d_{left} \\ \phi r_{right} &= d_{right}\end{aligned}$$

If $d_{baseline}$ is the distance between the left and right wheels, we can write:

$$r_{left} + d_{baseline} = r_{right}$$

Subtracting (2) from (3), we see:

$$\begin{aligned}\phi r_{right} - \phi r_{left} &= d_{right} - d_{left} \\ \phi(r_{right} - r_{left}) &= d_{right} - d_{left}\end{aligned}$$



$$\phi d_{baseline} = d_{right} - d_{left}$$

$$\phi = \frac{d_{right} - d_{left}}{d_{baseline}}$$

All of our arcs have a common origin at point P . Note that the angle of the robot's baseline with respect to the x axis is $\theta - \pi/2$. We now compute the coordinates of P :

$$\begin{aligned} P_x &= x - r_{center} \cos(\theta - \pi/2) \\ &= x - r_{center} \sin(\theta) \\ P_y &= y - r_{center} \sin(\theta - \pi/2) \\ &= y + r_{center} \cos(\theta) \end{aligned}$$

Now we can compute x' and y'

$$\begin{aligned} x' &= P_x + r_{center} \cos(\phi + \theta - \pi/2) \\ &= x - r_{center} \sin(\theta) + r_{center} * \sin(\phi + \theta) \\ &= x + r_{center} [-\sin(\theta) + \sin(\phi) \cos(\theta) + \sin(\theta) \cos(\phi)] \end{aligned}$$

And

$$\begin{aligned} y' &= P_y + r_{center} \sin(\phi + \theta - \pi/2) \\ &= y + r_{center} \cos(\theta) - r_{center} \cos(\phi + \theta) \\ &= y + r_{center} [\cos(\theta) - \cos(\phi) \cos(\theta) + \sin(\theta) \sin(\phi)] \end{aligned}$$

If ϕ is small (as is usually the case for small time steps), we can approximate $\sin(\phi) = \phi$ and $\cos(\phi) = 1$. This now gives us:

$$\begin{aligned}
x' &= x + r_{center}[-\sin(\theta) + \phi \cos(\theta) + \sin(\theta)] \\
&= x + r_{center}\phi \cos(\theta) \\
&= x + d_{center}\cos(\theta)
\end{aligned}$$

and

$$\begin{aligned}
y' &= y + r_{center}[\cos(\theta) - \cos(\theta) + \phi \sin(\theta)] \\
&= y + r_{center}[\phi \sin(\theta)] \\
&= y + d_{center}\sin(\theta)
\end{aligned}$$

In summary, our odometry equations for (x', y', θ') reduce to:

$$\begin{aligned}
d_{center} &= \frac{d_{left} + d_{right}}{2} \\
\phi &= \frac{d_{right} - d_{left}}{d_{baseline}} \\
\theta' &= \theta + \phi \\
x' &= x + d_{center}\cos(\theta) \\
y' &= y + d_{center}\sin(\theta)
\end{aligned}$$

How do we calculate dr, dl ?

Using wheel encoder sensor mounted behind each wheel. Each wheel encoder is used to count the number of times the motor (left or right) has rotated. This can be used to calculate the distance that the robot has driven or turned.

$\Delta tick_r = \text{currentTickCountRight} - \text{previousTickCountRight}$

$\Delta tick_l = \text{currentTickCountLeft} - \text{previousTickCountLeft}$

$dl = 2\pi R(\Delta tick_l / N)$

$dr = 2\pi R(\Delta tick_r / N)$

Where R is the radius, N number of opening slots in the encoding disk, and D is either dl or dr with a respective $\Delta tick$.

[View Code at Appendix A.1](#)

Chapter 3 Mechanical Structure and Hardware Design

3.1 Wheels and Chassis

The Robot have three wheels in total. Two of the three wheels are on the of center axis of the chassis. The third one will be a caster wheel, which helps for balancing the mass load of the robot. The two-navigation wheel are driven using a DC motor. The DC motor is combined with gears to increase the torque. The specification of the DC motor are as follow [9]:

- Operating voltage: 3V - 12V
- Max Torque: 800g/cm @ 3V
- Gear ratio: 1:48
- Load Current; 70mA(250mA max. @3V)
- Stall Current: 500mA max
- Weight: 29g
- Body size: 65mm x 37mm x 22mm (LxWxH)

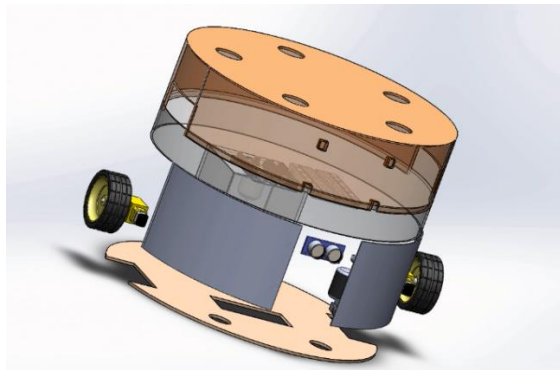


Figure 3:1 Solid work design exploded view of reclean

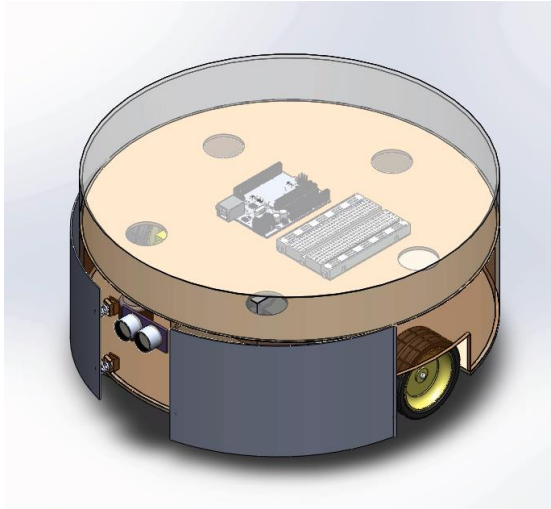


Figure 3:2 Solid work design 3D view of RoboClean

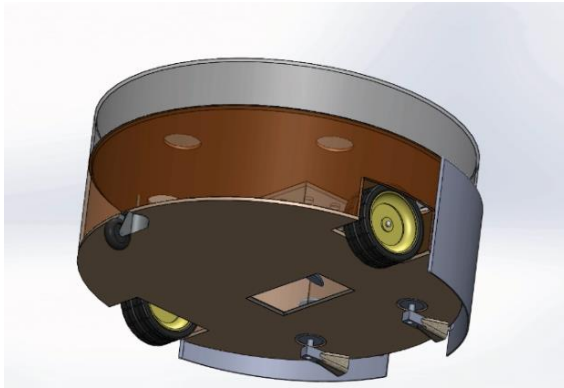


Figure 3:3 Solid work design 3D view captured from bottom of RoboClean

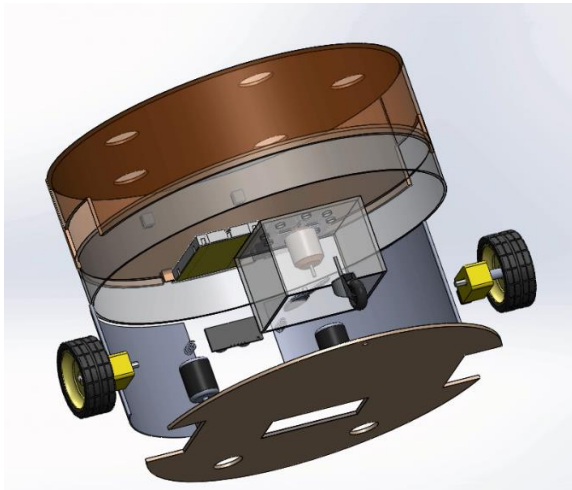


Figure 3:4 Solid work design 3D exploded view

3.2 Cleaning Structure

The robot is designed to be equipped with two rotating sweeper motors, each equipped with two brushes which spin the dirt into the vacuum region. The sweeper motor will be placed slightly away from the center axis. At the center of the robot there will be a vacuum region which will be created by the vacuuming fan motor. Figure 3.6 shows the structure of the dust bin. The dust bin is designed to have a fastest air flow at the bottom opening and has a place to mount fan motor at the top of the structure.

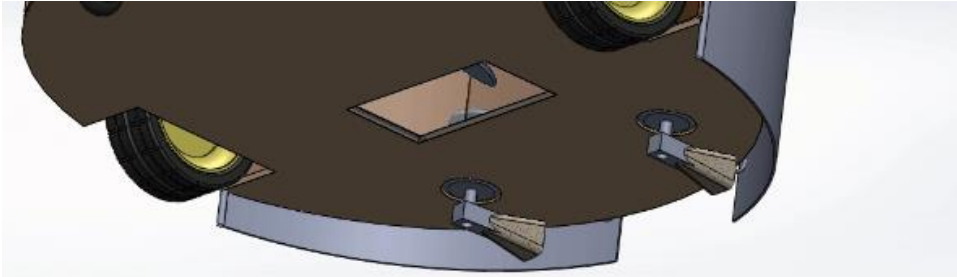


Figure 3:5 Cleaning section of RoboClean

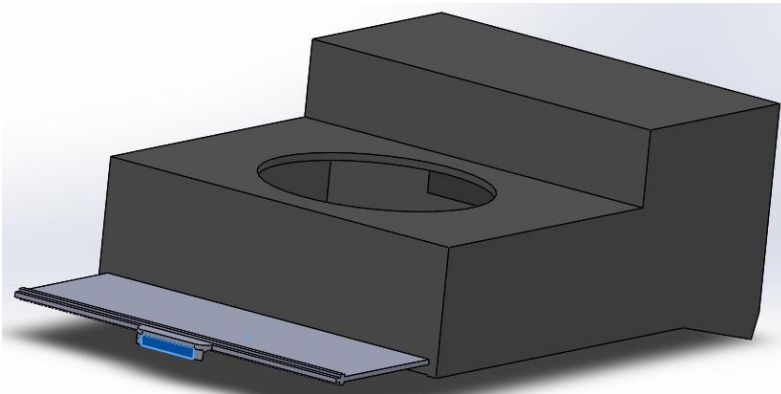


Figure 3:6 Dust bin with a removable plate to remove the dust

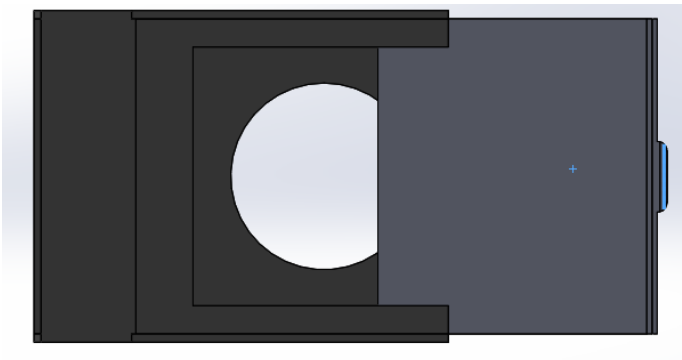


Figure 3:7 Bottom view of the dust bin while the removable plate moving

3.3 Perception

The robot needs sensors to get at least the gist of what the environment looks like and make its next action. There are various kind of range sensor which can be used for this project. In this project I will use a servo motor and an ultrasonic sensor together to measure the distance as shown on the figure.

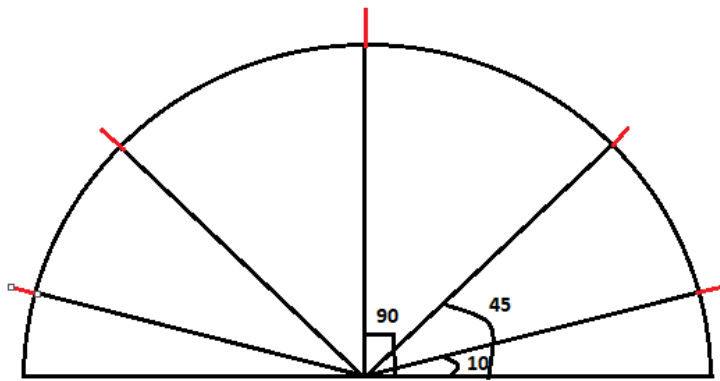


Figure 3:8 Distance measurement angle using servo and ultrasonic

The servo moves to the specified angle(10, 45, 90, 135, 170) and the ultrasonic will take the measurement.

Ultrasonic distance measurement module


HC-SR04 module will be used in this project. The following specification are for the module and are from its datasheet.

- Working Voltage DC 5V
- Working Current 15mA
- Working Frequency 40Hz
- Max Range 4m
- Min Range 4cm
- Measuring Angle 15 degree
- Trigger Input Signal 10us TTL pulse
- Echo pin outputs a pulse 150us to 25ms
- Echo pulse will timeout after 38ms of no object detected.
- At 20 degree Celsius the speed of sound is 343m/s

(from Elegoo datasheet for HC-SR04 module)

The time it takes for an HC-SR04 to measure distance:

Time a: time for starting up the trigger

$T_a =$  $= 13\mu s$

Time b: time for eight burst pulse using 40KHz

$$T_b = 8 \times \left(\frac{1}{40}\right) KHz = 0.2ms$$

Time c: time for the sound signal get back to receiver

T_{cn} : T_c for 4 cm

$$T_{cn} = 2 \times \left(\frac{4cm}{\left(\frac{34300cm}{s}\right)}\right) = 0.23ms$$

$T_{cf} = T_c$ for 4 meters

$$T_{cf} = 2 \times \left(\frac{4m}{\left(\frac{343m}{s}\right)}\right) = 23.4ms$$

Total time:

$$T_{max} = 13\mu s + 0.2ms + 23.4ms = 23.7ms$$

$$T_{min} = 10\mu s + 0.2ms + 233\mu s = 0.5ms$$

In one second the ultrasonic sensor can measure a minimum of $\frac{1}{23.7ms} = 42$ measurements.

How long does it take to measure all the five angular position?

The servo motor I am using, SG90, takes 0.12 sec / 60 degrees. Therefore, time it takes one measurement is the sum of time it takes for the servo motor to adjust it position, 0.1 sec, and time it takes for the ultrasonic sensor to measure 5 distinct distances, $5 \times 23.7ms = 118ms$;

$$\text{Time for one measurement} = 100ms + 118ms = 218ms$$

This implies in one second, we can measure 5 angular positions(10, 45, 90, 135,170 degree).

3.4 Control sensors and microcontroller

Arduino MEGA 2560

- Arduino Mega 2560 is a Microcontroller board based on Atmega2560. It comes with more memory space and I/O pins as compared to other boards available in the market.
- There are 54 digital I/O pins and 16 analog pins incorporated on the board that make this device unique and stand out from others.
- This board comes with two voltage regulators i.e., 5V and 3.3V which provides the flexibility to regulate the voltage as per requirements

[10]Arduino Mega is specially designed for the projects requiring complex circuitry and more memory space. Most of the electronic projects can be done pretty well by other boards available in the market which make Arduino Mega uncommon for regular projects. However, there are some projects that are solely done by Arduino Mega like making of 3D printers or for robotics controlling more than one motors, because of its ability to store more instructions in the code memory and a number of I/O digital and analog pins. Therefore, this project is implemented using Arduino Mega Micro Controller.

Node MCU ESP8622 development board

[11]The development board equips the ESP-12E module containing ESP8266 chip having Tensilica Xtensa® 32-bit LX106 RISC microprocessor which operates at 80 to 160 MHz adjustable clock frequency and supports RTOS.

[11]There's also 128 KB RAM and 4MB of Flash memory (for program and data storage) just enough to cope with the large strings that make up web pages, JSON/XML data, and everything we throw at IoT devices nowadays.

[11]The ESP8266 Integrates 802.11b/g/n HT40 Wi-Fi transceiver, so it can not only connect to a Wi-Fi network and interact with the Internet, but it can also set up a network of its own, allowing other devices to connect directly to it. This makes the ESP8266 NodeMCU even more versatile.

The feature explained above make Node MCU ESP8266 the perfect choice for controlling the robot from anywhere in this world. In this project the Node MCU ESP8266 works as a soft Access point and hosts a server.

Soft Access Point (AP) Mode

The ESP8266 that creates its own Wi-Fi network and acts as a hub (Just like Wi-Fi router) for one or more stations is called Access Point (AP). Unlike Wi-Fi router, it does not have interface to a wired network. So, such mode of operation is called Soft Access Point (soft-AP). Also, the maximum number of stations that can connect to it is limited to five. In AP mode ESP8266

creates a new Wi-Fi network and sets SSID (Name of the network) and IP address to it. With this IP address, it can deliver web pages to all connected devices under its own network. [12]

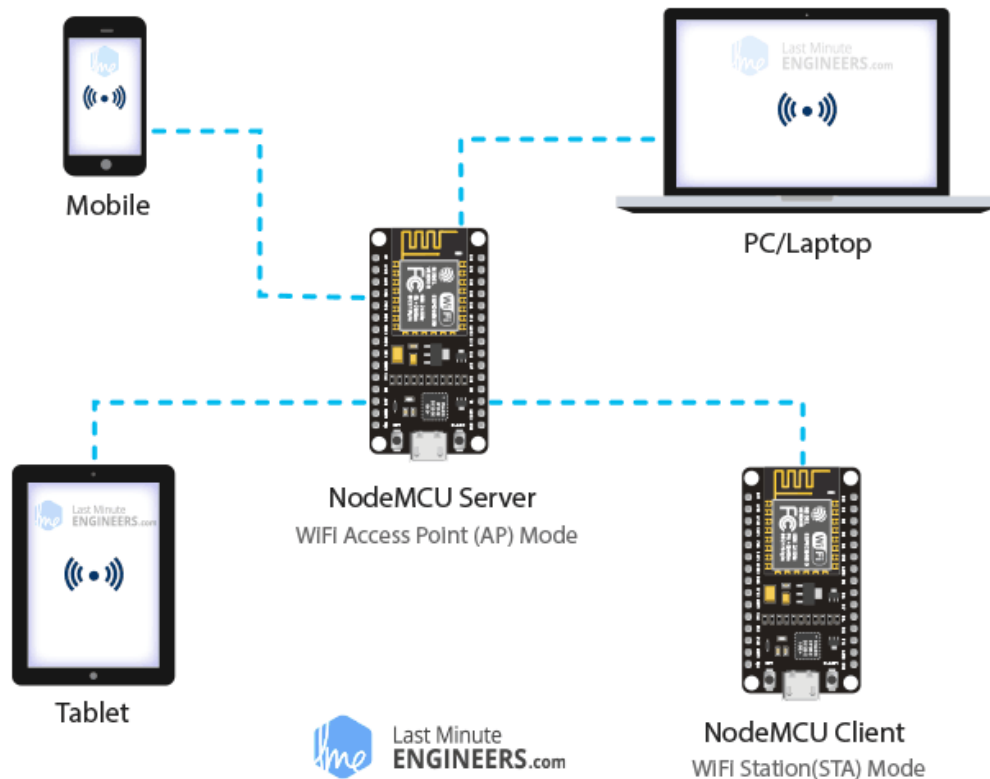


Figure 3:9 NodeMCU operating as an access point(Source:LastMinuteEngineering.com)

Ultrasonic Sensor

An ultrasonic sensor is an electronic device that measures the distance of a target object by emitting ultrasonic sound waves, and converts the reflected sound into an electrical signal. Ultrasonic waves travel faster than the speed of audible sound (i.e., the sound that humans can hear). Ultrasonic sensors have two main components: the transmitter (which emits the sound using piezoelectric crystals) and the receiver (which encounters the sound after it has travelled to and from the target) [13].

In order to calculate the distance between the sensor and the object, the sensor measures the time it takes between the emission of the sound by the transmitter to its contact with the receiver. The formula for this calculation is $D = \frac{1}{2} T \times C$ (where D is the distance, T is the time, and C is the speed of sound ~ 343 meters/second).

Optical Encoder

An incremental optical encoder, F249 Infrared speed sensor, is used for the purpose of measuring the speed and position of the robot. This IR speed module sensor with the comparator LM393, we can calculate the speed of rotation of the wheels of our robot. If we place a ring gear that rotates attached to our wheel. It could also be used as an optical switch.

The basic operation of this sensor is as follows; If anything is passed between the sensor slot, it creates a digital pulse on the D0 pin. This pulse goes from 0V to 5V and is a digital TTL signal. Then with Arduino we can read this pulse.



Figure 3:10 F249 Infrared speed sensor used for measuring speed and position



Figure 3:11 Speed encoder disk

3.5 Power source

All needed power to the components is supplied by a rechargeable battery with a rated voltage between 7V and 11V. No component needed power is supplied using the microcontroller. This will help to supply the necessary needed power for the components without putting too much current load the microcontroller.

Table 3.1 Maximum current usage of all the component while power on.

Components	Running time (%)	Number of components	Max Current drawn(mA)
Microcontroller	100	1	50
Wheel motor	100	2	2*500
Fan motor	100	1	250
Sweeper motor	100	2	2*250
Servo motor	100	1	100
Ultrasonic sensor	100	1	20
Total			1920

A power bank is the a readily available power source for driving the motors and Arduino. As we can see from below image, we can use normal USB cable in conjunction with a power bank to power the circuit.



Figure 3:12 Power Bank

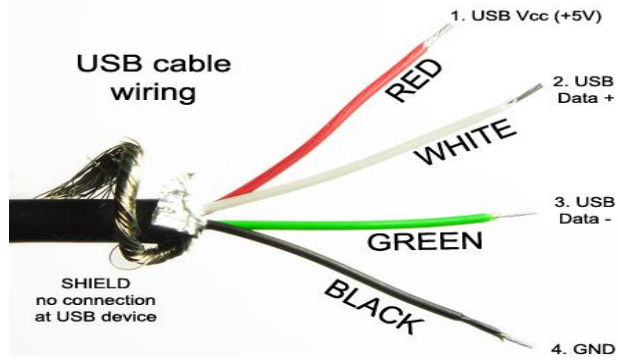


Figure 3:13 Inside the USB cable wire

Battery performance:

- Nominal Output current = 2.1A
- Nominal Output Voltage = 5 V
- Nominal Capacity = 10,000mAh.
- Total current consumption is 1920 mA
- Running time of the robot = $10,000 \text{ mAh} / 1920 \text{ mA} = 5.2 \text{ hour}$

3.6 Physical Dimension

Table 3.2 Physical dimension of the different parts of the robot

Components	Length(cm)	Width (cm)	height(cm)	Mass(g)
Dustbin	10	10	4	50
Chassis	25	25	-	200
Motor	1.8	7	7	20*5
Battery	7	14	1.5	200
breadboard	5.5	16.5	1	80
Arduino	6	11	1.5	40
Additional body part	-	-	-	200
Total				~900

3.7 Feasibility analysis

Power: considering the maximum total current(1920mA) that may be required by the robot, power source must deliver the required amount of current. The testing power bank has the capacity of 2.1A output current, which is enough to run the robot even at its peak power needed.

Torque: From the specification we can see that the starting torque of the motor is 800 g*cm. The robot is driven using two motor. Two motor, each having 800 g*cm torque, are expected to carry the whole mass around. The expected mass is calculated in the physical dimension table. The whole expected mass is around 900gram.

Chapter 4 Control System Design

4.1 Sampling time

There are some distinct tasks performed by the microcontroller. Driving the robot to the goal location, updating position and orientation state, sensing the environment for any obstacle, and checking for conditions to switch from one mode of operation to another. All these tasks have a clear priority from one another. For example, updating the pose state has a highest priority. This task should be implemented within a very small time period. Next to updating pose task, sensing the environment for an obstacle take the second priority over the rest of the tasks.

Since Arduino Mega has a limited processing speed of 16MHz, tasks should be given a different priority level from one another. This can be achieved by allocating a different minimum amount of time period at which the tasks must be performed.

- | | | |
|--|-------|-------|
| 1. Updating position and orientation | ----- | 100ms |
| 2. Sensing the environment | ----- | 150ms |
| 3. Checking condition to switch from one behavior to another | ----- | 500ms |

There is one other task which need an absolute high priority, which is processing signals generated from the wheel encoder. This task will be implemented using interrupt.

4.2 Design Architecture

As discussed in Chapter two the robot is model as differential drive with the equation as shown below:

$$\dot{x} = \left(\frac{V_r + V_l}{2}\right)\cos\phi$$

$$\dot{y} = \left(\frac{V_r + V_l}{2}\right)\sin\phi$$

$$\dot{\phi} = \frac{V_r - V_l}{L}$$

These equation shows that the input for the system are V_r and V_l . Controlling a robot by changing V_r and V_l seems counterintuitive therefore we will add another level of abstraction to the design. The new design method is to model the robot as a Unicycle and finally make a map between the new design method and the real implementation model design.

4.2.1 The Unicycle Model



Figure 4:1 Example of a unicycle drive

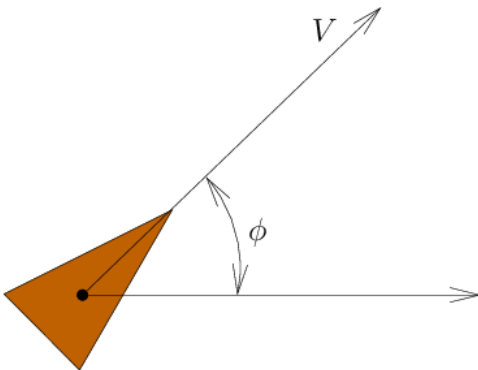


Figure 4:2 The Simplified Unicycle Robot Model

Inputs: Velocity (V) and angular velocity (w)

Dynamics

$$\dot{x} = V \cos \phi$$

$$\dot{y} = V \sin \phi$$

$$\dot{\phi} = \omega$$

This way we can control the robot using translational velocity and angular velocity if and only if we make a conversion from $(V, w) \rightarrow (V_l, V_r)$.

Design Model

Implementation model

$$\dot{x} = V \cos \phi$$

$$\dot{x} = \left(\frac{V_r + V_l}{2}\right) \cos \phi$$

$$\dot{y} = V \sin \phi$$

$$\dot{y} = \left(\frac{V_r + V_l}{2}\right) \sin \phi$$

$$\dot{\phi} = \omega$$

$$\dot{\phi} = \frac{V_r - V_l}{L}$$

Using these two sets of equation we can get the relation from (V, w) to (V_l, V_r) .

$$Vr = \frac{2V + \omega L}{2}$$

$$Vl = \frac{2V - \omega L}{2}$$

[View code at appendix A.2](#)

The above result implies we can have two models, one for design and the other for implementation and we can relate their input as shown on the above equations.

4.2.2 High level model

Controlling the robot using translational velocity and angular velocity makes a huge simplification from the implementation model, differential drive model. There are still questions that this model could not answer easily. For example, how could we determine the speed of the robot with respect to the goal location? Is this model helpful to design various behavior easily?

The above questions are not simply answered by the unicycle model alone. It would be very useful if we add an additional abstraction layer. I call this a high-level model.

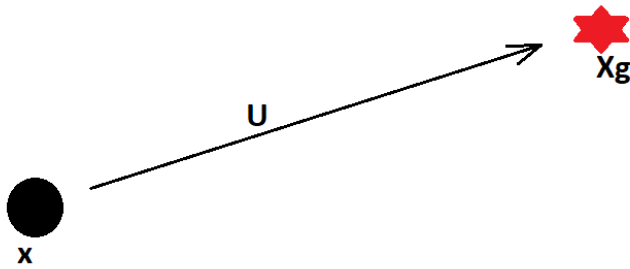


Figure 4:3 A robot and the control vector

Dynamics

$$\dot{x} = u, \quad x \in \mathbb{R}^2 \quad u = [u_x, u_y] \quad \dot{x} = [x_1, y_1]$$

This equation should be mapped to unicycle model of the robot. Fortunately, these two equations are mapped easily as shown below.

$$\phi_d = \text{atan}\left(\frac{u_y}{u_x}\right)$$

$$V = \sqrt{u_y^2 + u_x^2}$$

$$\omega = \text{PID}(e) = K_p e + K_i \int_0^\tau e(t) d\tau + \frac{K_d e(t)}{dt} \quad e = \phi_d - \phi$$

4.3 Go-to-Goal behavior

This is one of the most necessary behavior for a robot is to move from current location to a goal location, go-to-goal behavior.

$$e = x_g - x$$

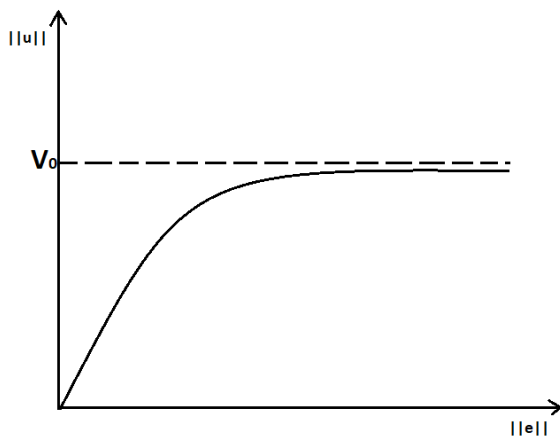
$$u = ke$$

$$\dot{x} = u$$

$$\dot{e} = 0 - \dot{x}$$

$$\dot{e} = -ke$$

A linear controller means the robot goes faster when it is further away from the goal. If we make the gain k a function of e , we can limit the amount of speed.



$$k = \frac{v_0(1 - e^{-\alpha\|e\|^2})}{\|e\|}$$

Figure 4:4 function used to regulate the value of u using the value of e

[View code at appendix A.3](#)

4.4 Obstacle avoidance

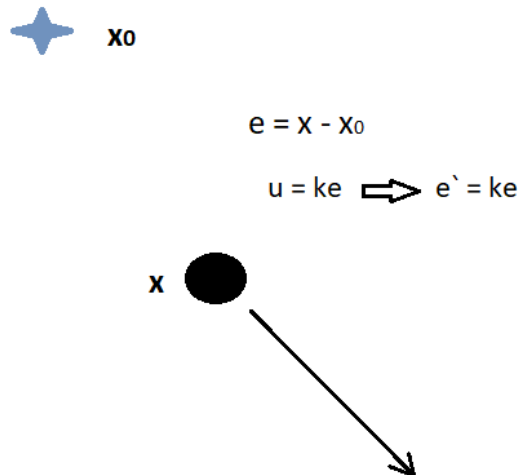


Figure 4:5 Robot's motion reaction near to obstacles

The error dynamics shows that the error is unstable and the robot will drive off to infinity. A simple solution is to switch to another behavior when the distance is greater than a given value.

In chapter three section 3.3 I have discussed about how the robot perceive the environment. It uses one servo motor with an ultrasonic sensor. The ultrasonic sensor only measures the distance to the obstacles in its own reference frame. It is necessary to change the measurement from the ultrasonic sensor to the world frame of reference. It is possible to achieve this by following simple three steps.

1. Transform the ultrasonic sensor distance measurement to a point in the robot coordinate frame

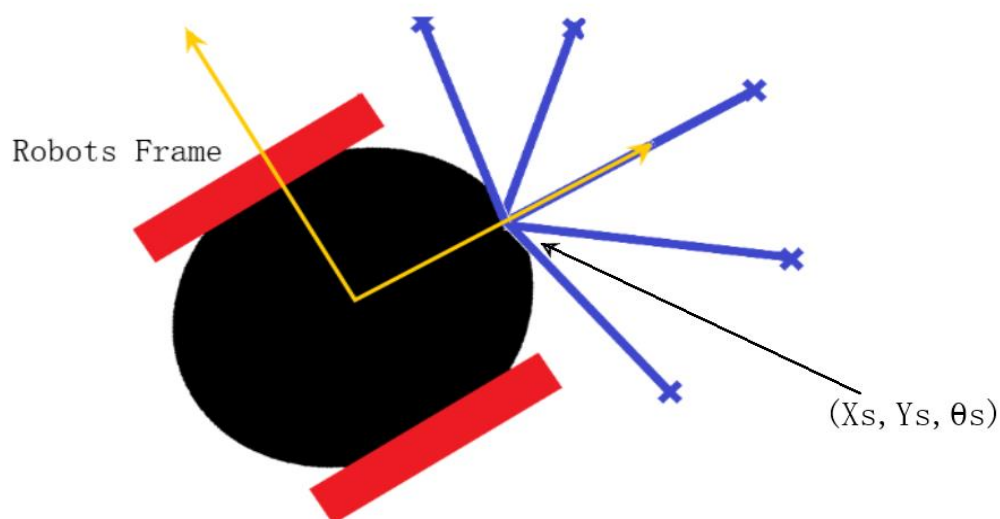


Figure 4:6 Sensor's coordinates are in the robot's frame

Transform the ultrasonic distance (which you converted from the raw IR values in Week 2) measured by each sensor to a point in the reference frame of the robot. A point p_i that is measured to be d_i meters away by sensor i can be written as the vector (coordinate) $v_i = [d_i, 0]$ in the reference frame of sensor i . We first need to transform this point to be in the reference frame of the robot. To do this transformation, we need to use the pose (location and orientation) of the sensor in the reference frame of the robot: $(X_{si}; Y_{si}; \theta_{si})$

Transformation matrix from the sensor(s) frame to the robot frame(r) T_s^r is used to transform sensor's coordinate to robot coordinate. θ_s in this case is zero degree.

$$T_s^r = \begin{bmatrix} \cos(\theta_s) & -\sin(\theta_s) & X_s \\ \sin(\theta_s) & \cos(\theta_s) & Y_s \\ 0 & 0 & 1 \end{bmatrix}$$

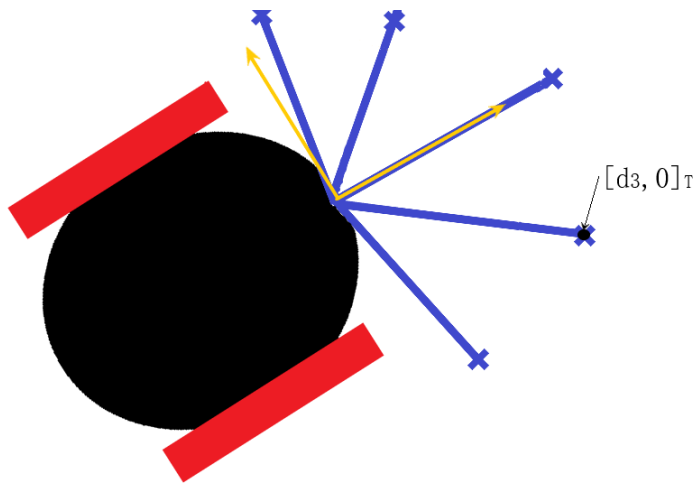


Figure 4:7 Ultrasonic distance defined in the sensor's frame

2. Transform the point from robot coordinate frame to the world coordinate frame

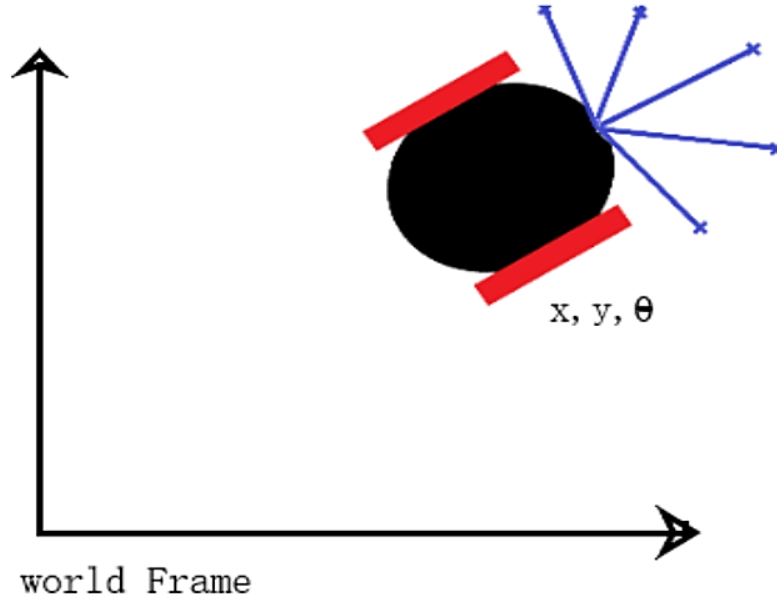


Figure 4:8 Robot's coordinates are in the world frame

A second transformation is needed to determine where a point p_i is located in the world that is measured by sensor i . We need to use the pose of the robot, $(x; y; \theta)$, to transform the robot from the robot's reference frame to the world's reference frame. This transformation is defined as:

$$T_r^o = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix}$$

Using T_r^o and T_s^r It is possible to find the position of the obstacle in the world frame coordinate as show below.

$$[x_o; y_o; 0] = T_r^o \times T_s^r \times [d_{xi}; d_{yi}; 0]$$

3. Compute a vector that points away from any obstacles

Use the set of transformed points to compute a vector that points away from the obstacle. The robot will steer in the direction of this vector and successfully avoid the obstacle. The vector that points away from the obstacle can be found using the robot position and the obstacle position. Weigh each vector according to their importance, $\alpha_1 u_1, \alpha_2 u_2, \dots, \alpha_9 u_9$. For example, the front and side sensors are typically more important for obstacle avoidance while moving forward.

$$u_{ao} = [x; y] - [x_o; y_o]$$

Chapter 5 Testing, Simulation Result, and Conclusion

5.1 Result

The project is composed of various problem area. It is composed of working with electrical devices, designing and implementing a mechanical structure, programming microcontroller and working with web development for user interface and configuration. The following sub sections will discuss the results on their distinctive problem area.

5.1.1 Implementation of Control

As discussed in chapter 4, the way the robot is controlled is by dividing the tasks into different behaviors. I have implemented Go to Goal and Obstacle avoidance behavior. These two-behaviors are able to tackle about 80 percent of the robot navigation. It can navigate in any convex environment without stacking into traps. However, in order to navigate in concave environment, an additional wall following behavior is needed, which I didn't implement due to time constraint.

Go To Goal behavior

Test One: Robot moving from position from $x = 0$ and $y = 0$ to goal position $x = 150$, and $y = 150$.



PID parameters: $K_p = 2$, $K_i = 0.05$, $K_d = 0$.

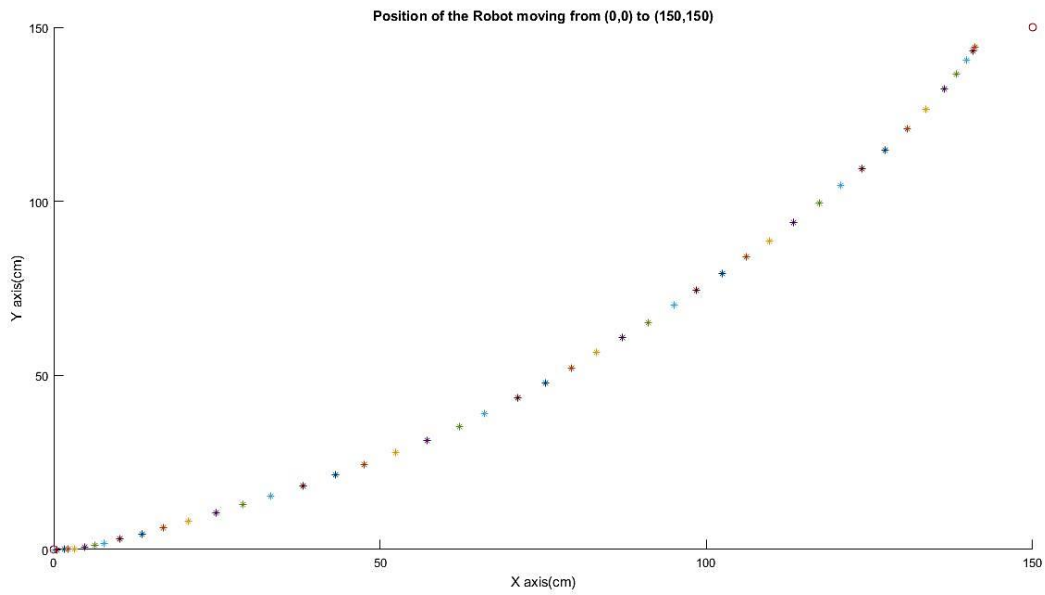


Figure 5:1 Position of Robot while moving from (0,0) to (150,150)

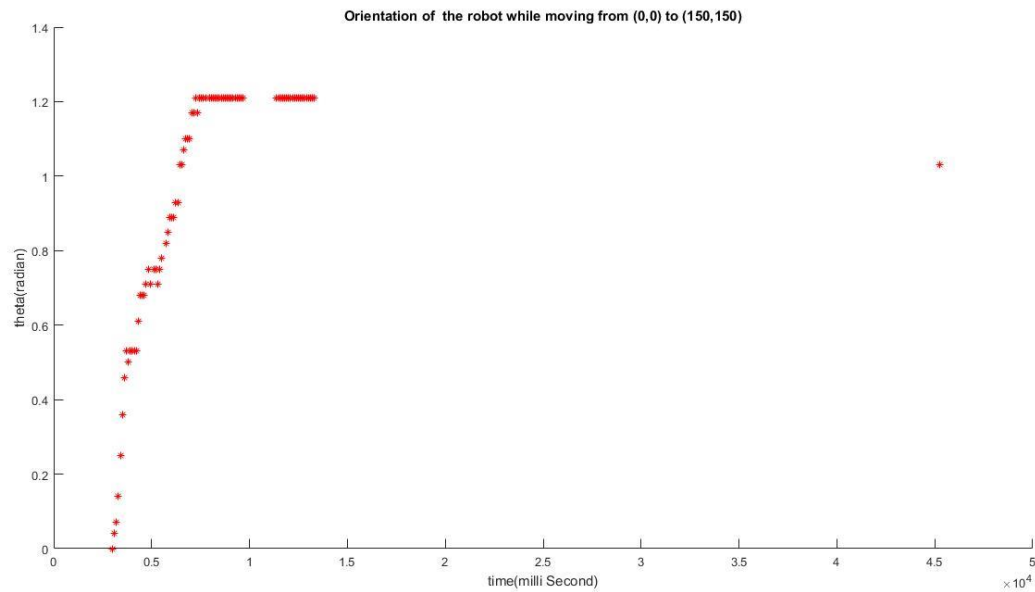


Figure 5:2 performance of the robot angle tracking while moving from position (0,0) to (150,150)

Test two: Robot moving a straight line from position from $x = 50$ and $y = 0$ to goal position $x = 150$, and $y = 0$

PID parameters: $K_p = 2$, $K_i = 0.05$, $K_d = 0$

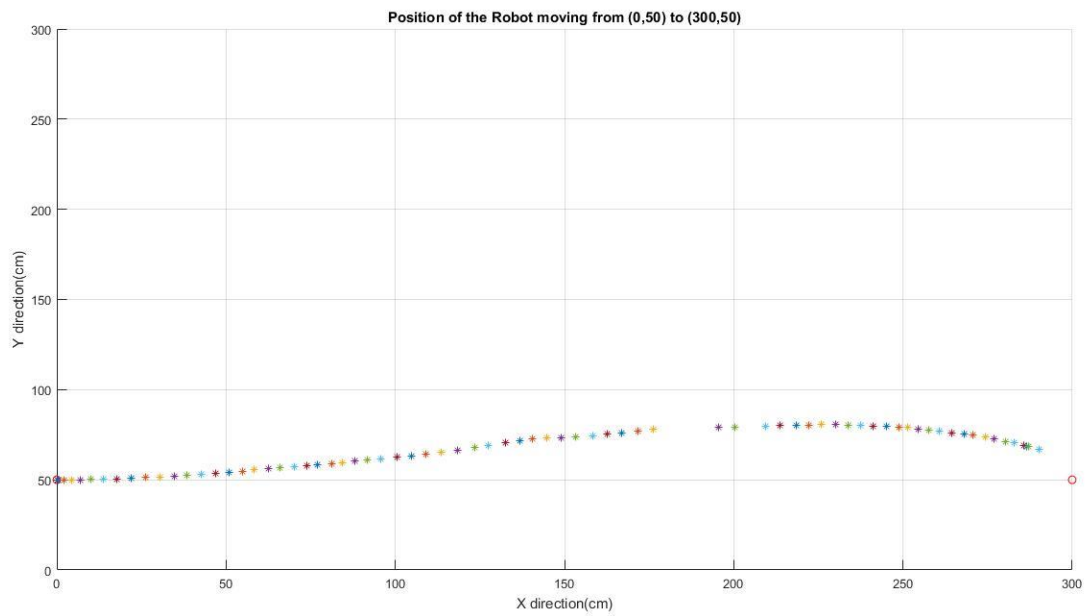


Figure 5:3 performance of the robot moving on a straight line for PID of $K_p = 2$, $K_i = 0.05$, $K_d = 0$

PID parameters: $K_p = 2.5$, $K_i = 0.5$, $K_d = 0.1$

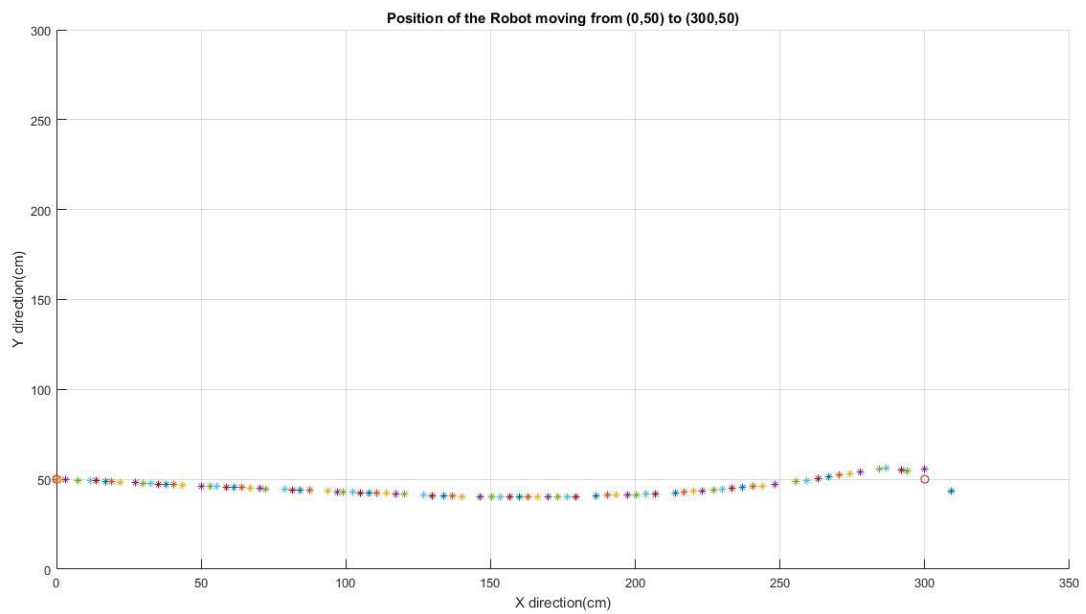


Figure 5:4 performance of the robot moving on a straight line for PID of $K_p = 2.5$, $K_i = 0.5$, $K_d = 0.1$

ID parameters of $K_p = 20$, $K_i = 10$, $K_d = 5$

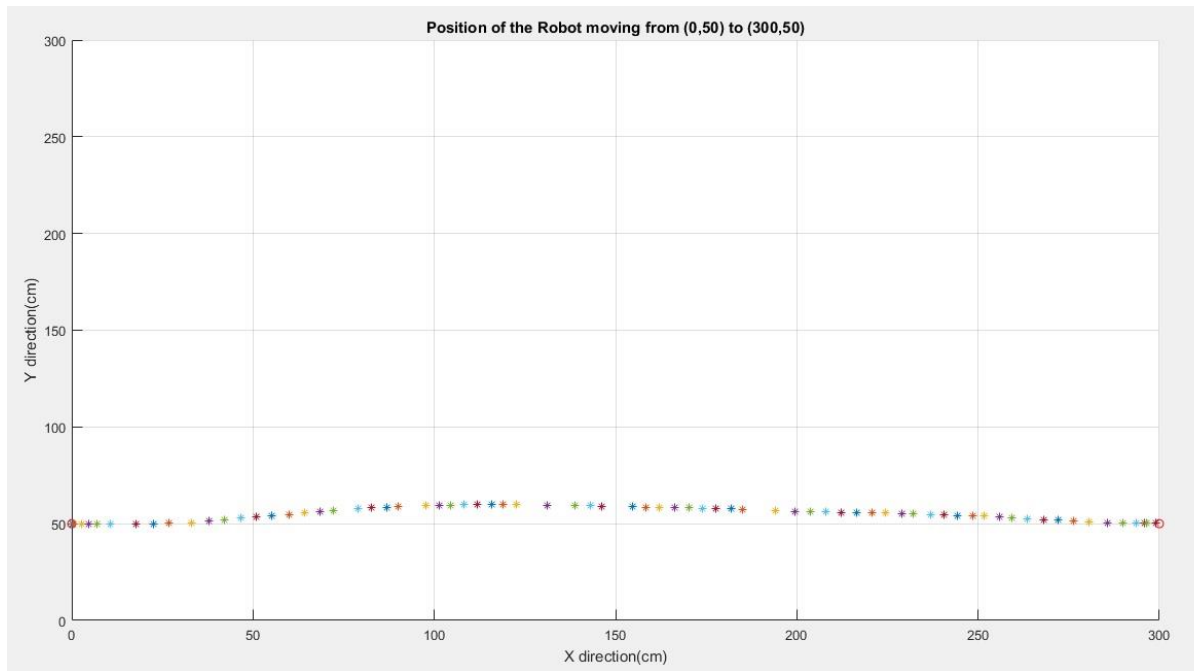


Figure 5:5 performance of the robot moving on a straight line for PID of $K_p = 20$, $K_i = 10$, $K_d = 5$

Figure 5.5 shows PID parameters' of $K_p = 20$, $K_i = 10$ and $K_d = 5$ give the best result for the robot while moving in a straight line.

5.1.2 Hardware

Trial One:

- Foam board for robot's chassis implementation
- Motor and Infrared code is written

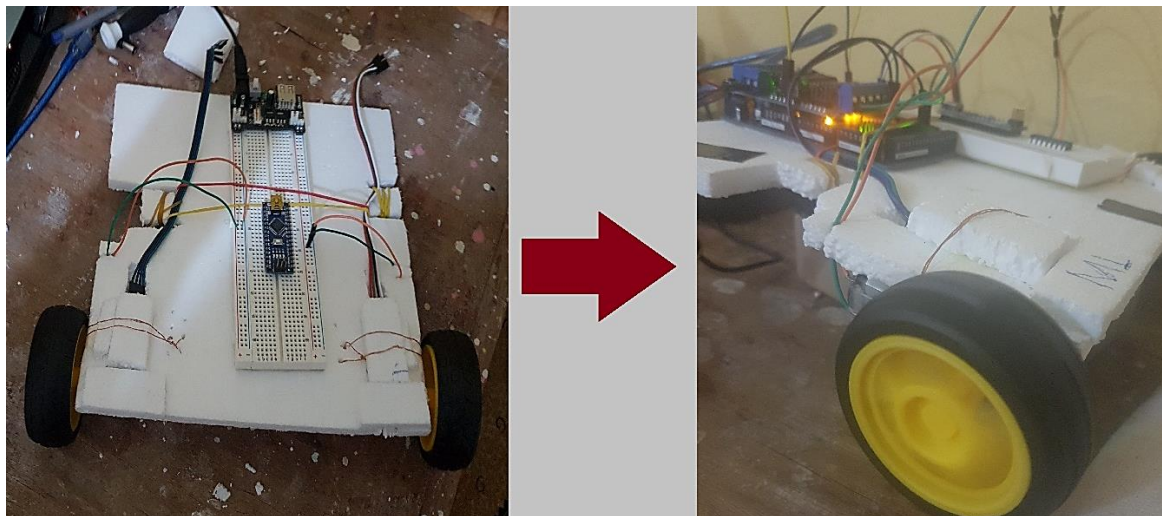


Figure 5:6 Phase one implementation of RoboClean

Trial two: upgraded to cardboard and most of the code is written except ultrasonic sensor and servo motor code.

Chassis structure is upgraded to cardboard

Odometry is implemented

Go-To-Goal behavior is implemented

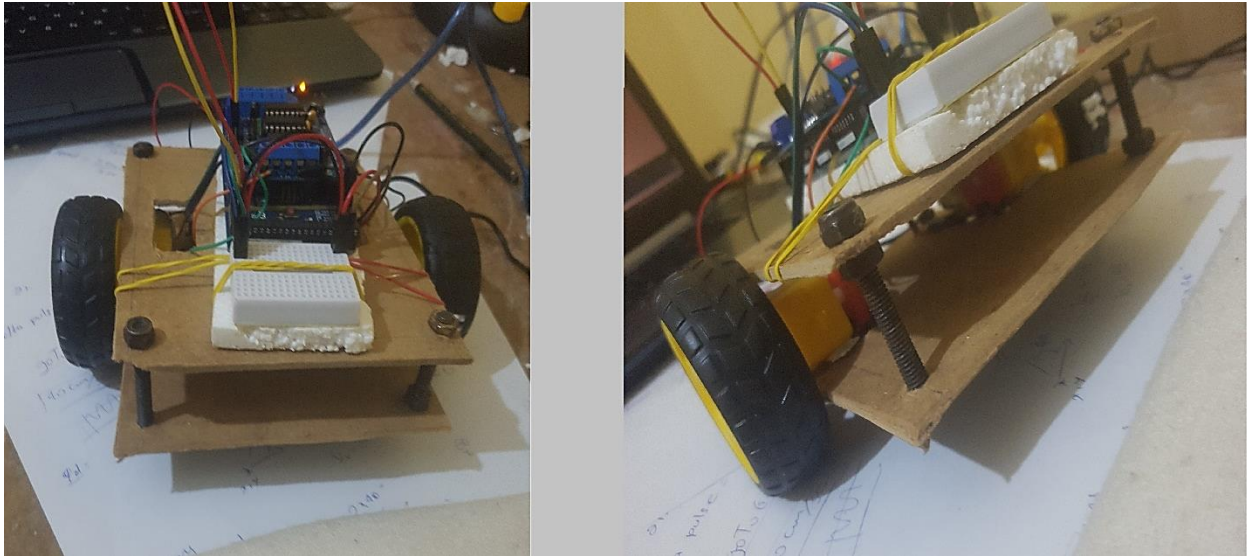


Figure 5:7 Phase two implementation of RoboClean

Trial three:

upgraded to designed actual size

ultrasonic and servo is added and implemented

Node MCU esp8622 W-Fi module is added and the robot is interfaced with a web UI



Figure 5:8 Phase three implementation of RoboClean

Last trial

Vacuum cleaning mechanism is implemented

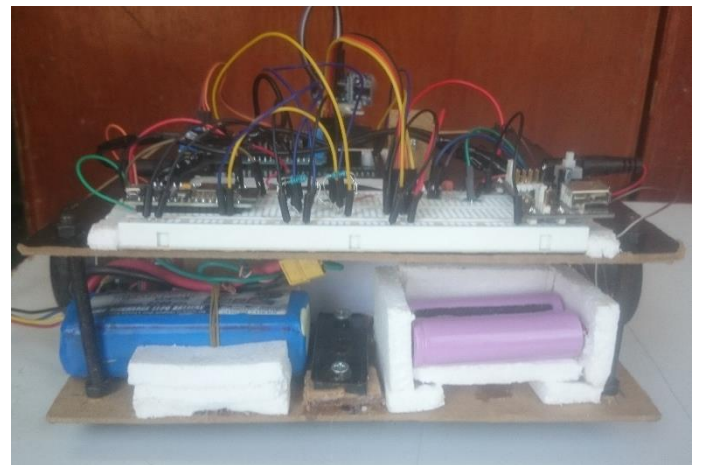
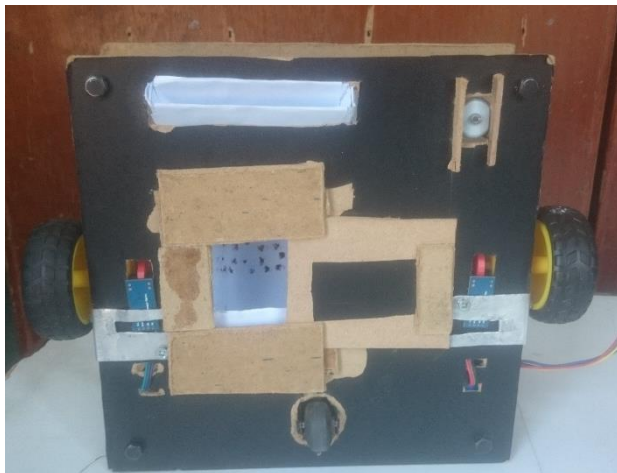
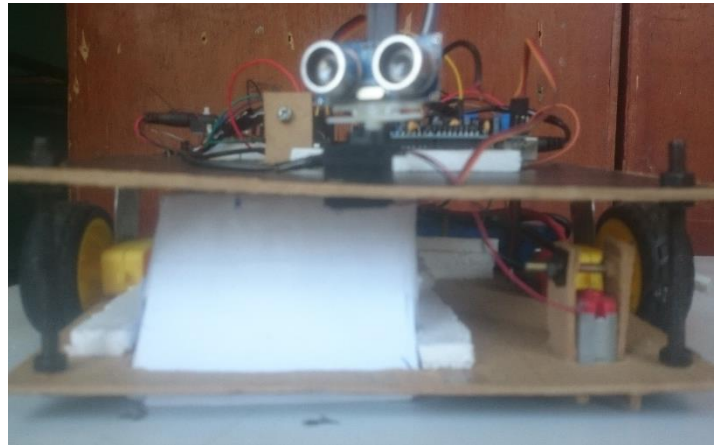
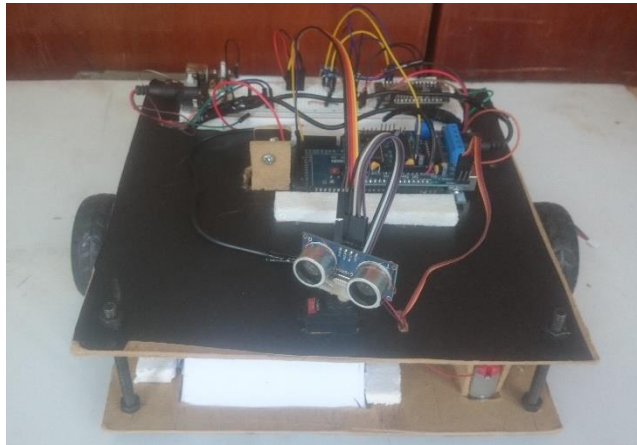


Figure 5:9 Last phase implementation of RoboClean

5.2 Conclusion

This paper highlights a better and simpler approach in providing an overview of the design of an autonomous vacuum cleaner robot. This robot is designed to have all the features of a conventional vacuum cleaner. It can work automatically and manually. The robot can be made useful in large areas of coverage such as hospitals, industries, schools, etc. It can also be used where cleaning involving humans may be harmful.

A disadvantage of odometry is that the measurements are indirect, relating the power of the motors or the motion of the wheels to changes in the robot's position. This are often error-prone since the relation between motor speed and wheel rotation can be very nonlinear and vary with time. It can provide good dead-reckoning over short distances, but error accumulates very rapidly. At every step, we introduce, error (noise) into not just x and y , but also θ . The error in θ is the worst, since every error in θ are going to be amplified by future iterations. There are a number of basic noise sources.

Slippage. When the robot turns a corner, one wheel will, generally, slip a little bit. This means that albeit the odometry data was perfect, the robot's path wouldn't recoverable from it.

Error in estimate of dbaseline or in wheel diameters. We must measure the baseline of the robot and therefore the multiplier relating angular rate and linear velocity (a function of the wheel diameter.) Even small errors in these constants can cause problems—typically, the odometry result will have a uniform veer in one direction or the opposite. In addition, the wheels on the robot are slightly different sizes and deflation of wheel also cause change the wheels to have different size before deflation. Even small differences cause to large navigation errors with time.

Improved estimates of position can be obtained by using an inertial navigation system, which directly measures acceleration and angular velocity that can be used to determine the robot's position

I would also argue spending a reasonable resource while buying the wheel motor. It is one of the most important part of the robot. Anything done in the robot is implemented by the wheel robot. In my case I was able to use the motor with some tricks. The challenge I faced on the wheel motor is moving the whole robot at low speed. At low speed the motor was not able to supply the required torque to move the robot. Even though I made a good decision buying geared motor with 1:48 gear ratio, I would recommend spending a reasonable amount of investment on the wheel motor.

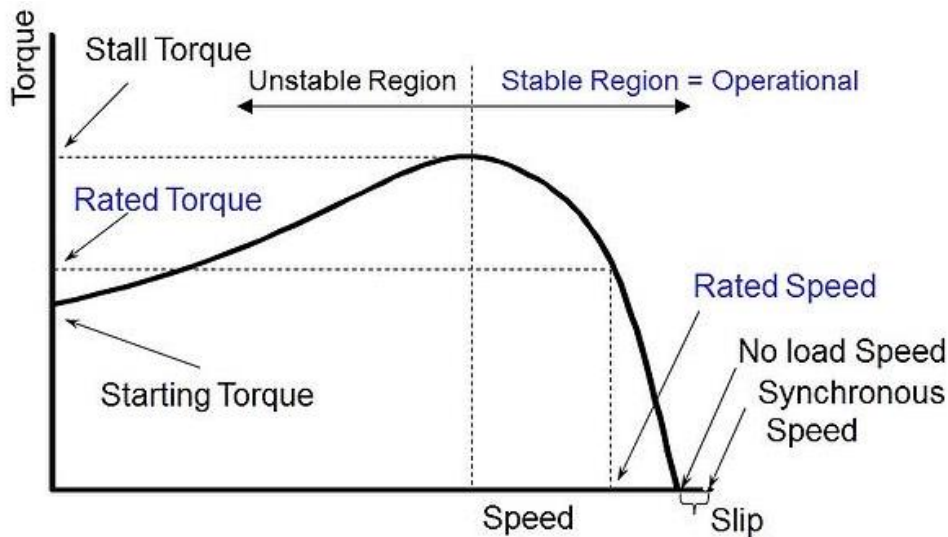


Figure 5:10 Torque for a given speed of a dc motors(courtesy: <https://blog.orientalmotor.com/eliminate-motor-speed-fluctuations-due-to-input-voltage-or-load-variance>)

There are some options which can be used for the robot chassis. Light metal sheet, strong cardboard, or various kinds of light weight plastics. I chose to work with cardboard. I found out that it works well for small size chassis. But increasing the size of chassis make the cardboard to be somewhat flexible and bendable which are obviously undesired.

There were some choices of sensors, for the perception implementation, on the table. One can use Camera, LIDAR, IR sensor and Ultrasonic sensor. Each of the sensor has their distinctive merit and also some weakness. Based on the price and simplicity I chose ultrasonic together with a servo motor to implement the robot's eye. Even though It works fairly for small scale project, I would recommend to go with either Camera or LIDAR.

Chapter 6 References

- [1] T. Asafa, "Development of a vacuum cleaner robot," p. 1.
- [2] "Robot Platform Theory," [Online]. Available:
http://www.robotplatform.com/knowledge/Classification_of_Robots/wheel_control_theory.html.
- [3] S. M. K. K. A. Y. A. Z. A. S. K. a. F. S. R Visvanathan, "Mobile robot localization system using multiple ceiling mounted cameras," pp. 1-4.
- [4] T. S. a. Y. Kuroda, "Mobile robot localization by gps and sequential," pp. 25-30, 2013.
- [5] M. G. B. a. E. M. Dariush Forouher, "Sensor fusion of depth camera and ultrasound data for obstacle detection and robot navigation," *IEEE*, pp. 1-6, 2016.
- [6] R. S. Sanketh, "Differential Drive," 23 May 2020. [Online]. Available:
<https://medium.com/manual-robotics/differential-drive-4b166ad736a6>.
- [7] D. a. Jenkin, in *Computational Principles of Mobile Robotics*, 2000.
- [8] E. Olson, in *A Primer on Odometry and Motion control*.
- [9] "Wiltronics," [Online]. Available: <https://www.wiltronics.com.au/product/10137/yellow-motor-3-12vdc-2-flats-shaft/>.
- [10] S. Z. Nasir, "Introduction to Arduino Mega," [Online]. Available:
] <https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-mega-2560.html>.
- [11] L. m. engineering, "Insight into ESP8266 NodeMCU," [Online]. Available:
] <https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/>.
- [12] L. M. Engineering, "Last Minute Engineering," Last Minute Engineering, 2018. [Online].
] Available: <https://lastminuteengineers.com/creating-esp8266-web-server-arduino-ide/>.
- [13] D. Jost, "What is Ultrasonic Sensor," [Online]. Available:
] <https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor#:~:text=An%20ultrasonic%20sensor%20is%20an,sound%20that%20humans%20can%20hear>.

Chapter 7 Appendix A

7.1 Odometry implementation in Arduino

```
void odometry()
{
    Dl = ((float)(currentLeftEncoderPulses - previousLeftEncoderPulses) / numberOfHole) * PI * diameter;
    previousLeftEncoderPulses = currentLeftEncoderPulses;
    Dr = ((float)(currentRightEncoderPulses - previousRightEncoderPulses) / numberOfHole) * PI * diameter;
    previousRightEncoderPulses = currentRightEncoderPulses;
    byte i = 0;
    // mean filter
    for (i; i < ARRAYSIZE - 1; i++)
    {
        arrDl[i] = arrDl[i + 1];
        arrDr[i] = arrDr[i + 1];
    }
    arrDl[i] = Dl;
    arrDr[i] = Dr;
    i = 0;
    for (i; i < ARRAYSIZE; i++)
    {
        meanDl += arrDl[i];
        meanDr += arrDr[i];
    }
    meanDl /= ARRAYSIZE;
    meanDr /= ARRAYSIZE;

    Dc = ((float)meanDl + meanDr) / 2;
    x = x + Dc * cos(phi);
    y = y + Dc * sin(phi);
    phi = phi + (meanDr - meanDl) / L;
    phi = atan2(sin(phi), cos(phi));
}
```

7.2 Conversion between unicycle and differential drive model

```
// unicycle( v,w) to differential (Vl,Vr)
float Vl_d = (2*V - W*L)/2;
float Vr_d = (2*V + W*L)/2;
```

7.3 Go to Goal implementation

```
void goToGoal(){
    if(abs(Xg - x) < 5 && abs(Yg - y) < 5){
        moveMotor(LEFT_WHEEL, FORWARD, 0);
        moveMotor(RIGHT_WHEEL, FORWARD, 0);
        return;
    }
    phid = atan2(Yg-y, Xg-x);
    phiErr = phid - phi;
    phiErrSum += phiErr * delta/1000;
    if(phiErr < PI/8 && phiErr > -PI/8){
        smallPhiErrorController();
    }else{
        largePhiErrorController();
    }
}

float smallPhiErrorController(){ // for phiError < pi/8 or phiError > -pi/8
    // PID controller for angular velocity of the robot
    W = Kp * phiErr + Ki*phiErrSum + Kd*((phiErr-phiErrOld)/delta);
    phiErrOld = phiErr;

    // validate if the angular velocity is ensured
    ensure_W();

    leftMotorSpeed = map(Vl, robotMinSpeed, robotMaxSpeed, PWMmin, PWMmax);
    rightMotorSpeed = map(Vr, robotMinSpeed, robotMaxSpeed, PWMmin, PWMmax);
    moveMotor(LEFT_WHEEL, FORWARD, leftMotorSpeed);
    moveMotor(RIGHT_WHEEL, FORWARD, rightMotorSpeed);
}

float largePhiErrorController(){ // for phiError > pi/8 or phiError < -pi/8
    if(phiErr >= PI/8){
        // move the robot in counter clockwise
        moveMotor(LEFT_WHEEL, FORWARD, 0);
        moveMotor(RIGHT_WHEEL, FORWARD, PWMmin);
    }else if(phiErr <=-PI/8){
        // move the robot in clockwise
        moveMotor(LEFT_WHEEL, FORWARD, PWMmin);
        moveMotor(RIGHT_WHEEL, FORWARD, 0);
    }
}
```

```

    phiErrSum = 0;
    phiErrOld = 0;
}

void ensure_W(){
    // This function ensures that w is respected as best as possible
    // by sacrificing v.

    // 1. Limit v,w from controller to +/- of their max
    V = max(min(V ,Vmax_W0), -Vmax_W0);
    W = max(min(W, Wmax_V0), -Wmax_V0);
    // unicycle( v,w) to differential (Vl,Vr)
    float Vl_d = (2*V - W*L)/2;
    float Vr_d = (2*V + W*L)/2;
    // Find the max and min
    float V_rl_max = max(Vr_d, Vl_d);
    float V_rl_min = min(Vr_d, Vl_d);

    // Shift vel_r and vel_l if they exceed max/min vel
    if(V_rl_max > Vmax){
        Vl = Vl_d - (V_rl_max - Vmax);
        Vr = Vr_d - (V_rl_max - Vmax);
    }else if(V_rl_min < -Vmax){
        Vl = Vl_d - (V_rl_min + Vmax);
        Vr = Vr_d - (V_rl_min + Vmax);
    }else {
        Vl = Vl_d;
        Vr = Vr_d;
    }
}
}

```

