AUTONOMOUS WATER SURFACE CLEANING BOT

By Prakhar Sah

TABLE OF CONTENT

1. Introduction

Climate change is real and it's happening faster than any of us could have imagined. Earth is our home planet, preserving life on it for our survival and in order to prevent the inadvertent extinction of all life forms on earth, we need to start taking measures. Water pollution is one of the biggest issues in India. Our water bodies have lost their pristine beauty over the years and now it seems absurd to think that's ever going to change. With the "object detecting autonomous water surface cleaning bot" (AWSC) we have tried to take a step in this direction.

The AWSC bot's primary purpose is to collect the solid waste floating on the water surface. It can also be used to clear the eutrophication layer which is commonly found on stagnant water. The bot utilizes its motion to capture the floating waste like a whale does in the ocean. It is autonomous which means no external control is required and the bot has all the necessary control systems to work on its own. It achieves this by using cost effective mapping and localization techniques to navigate itself in an unknown territory and solving the problem of global localization.

In this project our primary objective is to make a completely autonomous bot at a low budget. We have used wheel encoders for odometry along with Ultrasonic & IR sensors and push button sensors for mapping, localization and obstacle detection purposes, instead of using costly lidar sensors. The main navigation model of the bot is based on the differential drive mobile robot model, which we will explain in the subsequent sections.

The bot is designed to operate in still water bodies like lakes, ponds, swimming pools or algae infested stagnant waters. It serves its purpose efficiently by completing the following three stages:

- **Mapping:**
  For this we use the method of least squares and the coastal rule algorithm to compute a map of the unknown region, the water body in our case. After the map is built, we use computational geometry to set the goal points.

- **Localization:**
  For this we use the Monte Carlo Localization technique (particle filter sampling) which is a probabilistic approach to find the pose of the bot in the map. This filter works in conjunction with the odometry to continuously feed the navigation model with the pose of the robot.

- **Navigation Model:**
  The navigation model is based on the differential drive mobile robot model for tracking the motion of the bot towards the goal points as determined during the first stage. While it navigates across the water body, the AWSC bot captures floating solid waste through a net attached to it.

The entire code for the bot is written on MATLAB and Python which is then deployed on a Raspberry Pi 3B+ wirelessly from the host PC.

2. Mapping

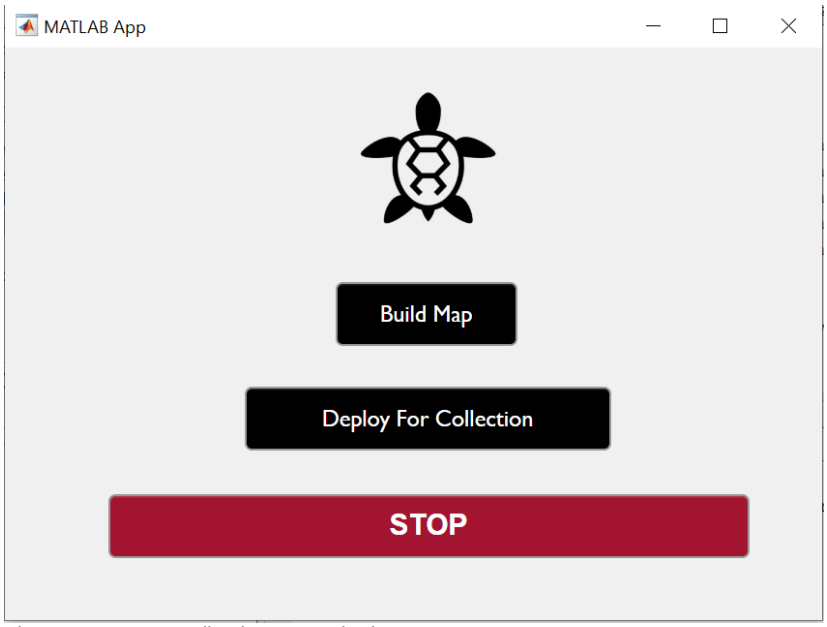

Fig 1 – MATLAB application to run the bot

The first stage before we can begin any kind of deployment for cleaning is to make a map of the unknown region so that the bot can localize itself as well as set the goal points for defining the motion of the bot.

For this purpose, we have built an app in MATLAB. Pushing any button will run the particular function assigned to it and relay the information to the Raspberry Pi by making a connection to the board's IP address over the wireless connection to which both the Pi and the PC must be connected.

Start by pressing the Build Map button. This will activate the coastal rule algorithm and the bot starts mapping the region. It will store this map in a 'map.mat' which can be accessed by the functions for the localization and navigation model.

The method of mapping is based on the coastal rule and method of least squares as explained in [1]. After the map is drawn, computational geometry is used to set the goal points for the navigation model.

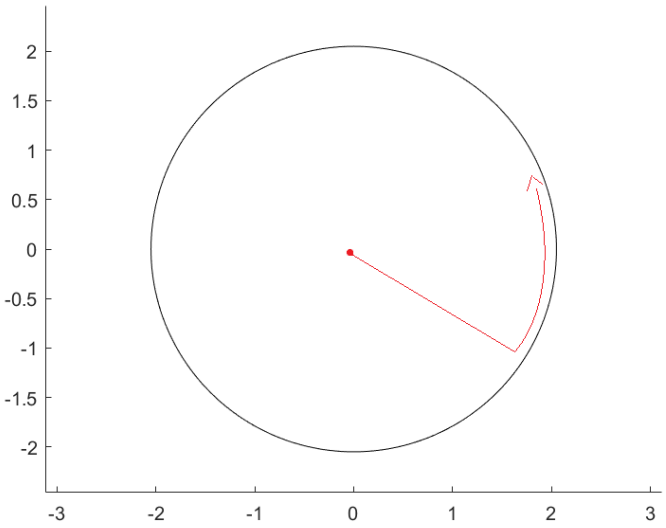

Fig 2. Motion of the bot inside water body (a circular water body in this example)

The starting point of the bot is set as the origin and odometry initiates. The bot will keep moving straight in its direction of heading until it locates a wall. Once a wall has been identified, the wall is tracked using the right IR sensor on the bot, which makes the bot follow the wall in an anti-clockwise direction, until it completes one full revolution around the boundary region. The [x, y] co-ordinates of the bot are stored as the bot makes this revolution.

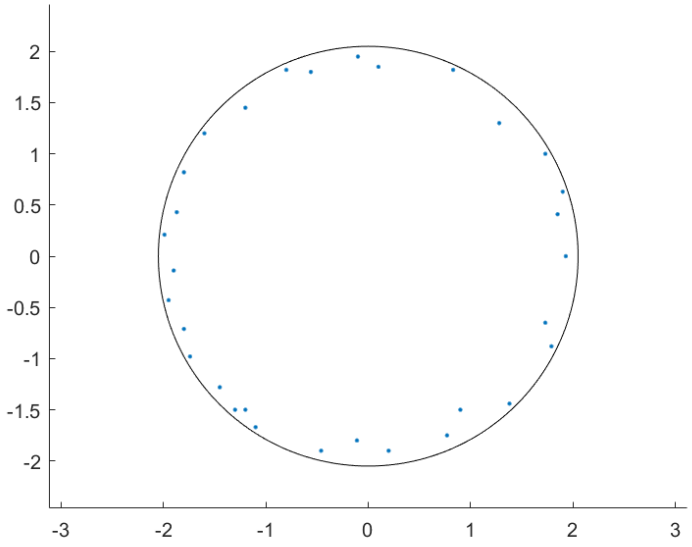

Fig 3 – Points stored in the ‘map.mat’ file

After these points are stored, a map is approximated using computational geometry.

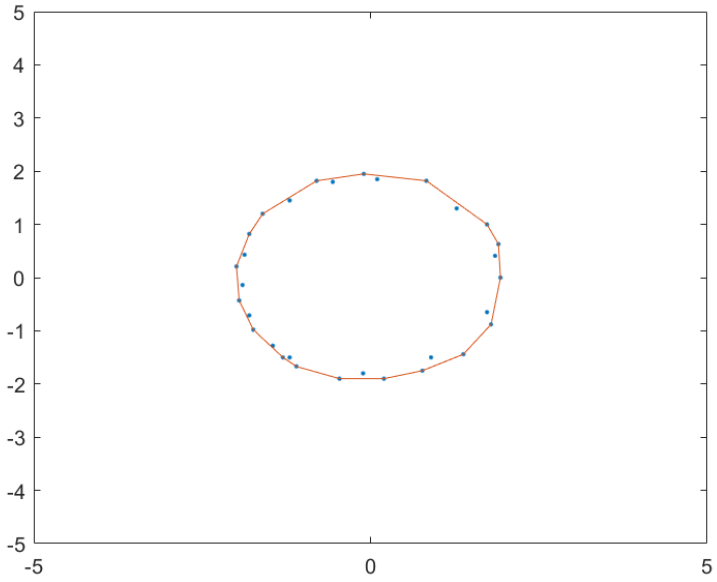

Fig 4 – Map made by the bot using coastal algorithm and computational geometry

Next the goal points are set on the computed map. We believe that a zig-zag pattern of movement would be the most efficient way to cover the whole region and maximize the waste collected by the bot. Hence, we set the goal points accordingly.

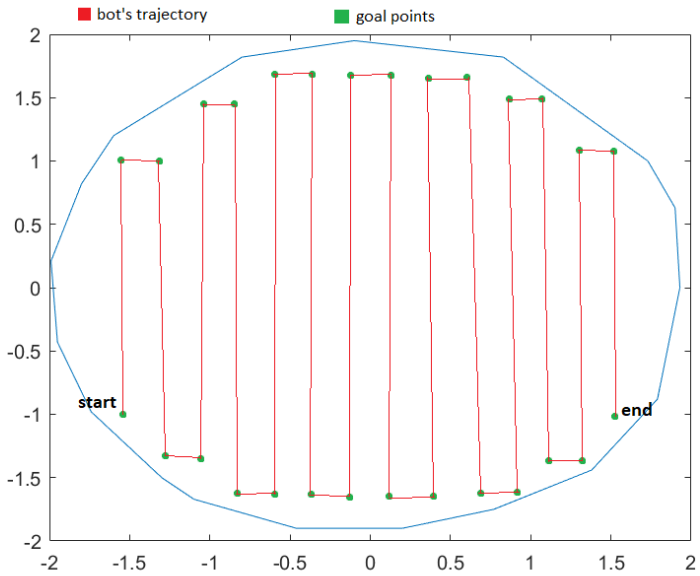

Fig 5 – Goals points in green, desired bot trajectory in red

Now that the map is computed and goal points set, we can skip building the map again next time we operate the bot in the same water body, and can directly deploy the bot for cleaning.

3. Localization

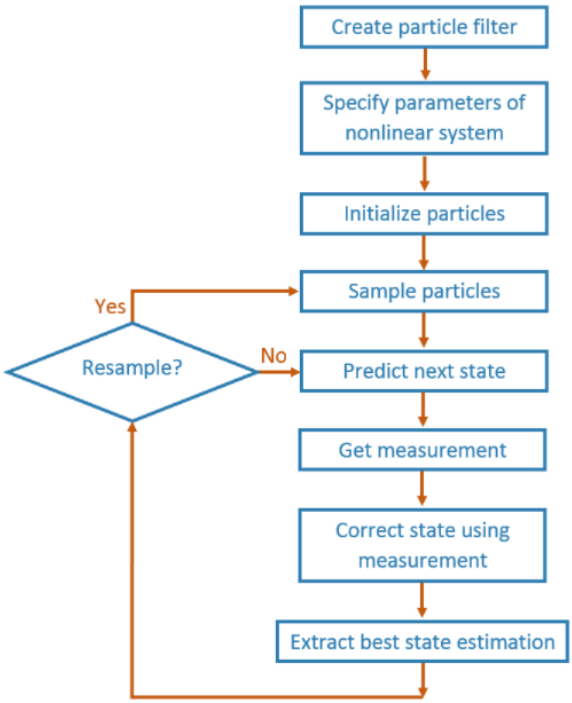

Fig 6 – General workflow of a particle filter

We use a particle filter for estimating the pose [x, y, θ] of the bot. The particle filter samples the readings from the odometry values provided by the wheel encoders and uses the map and ultrasonic sensor readings to give the probability of the bot being located at different points of the map. This way we can estimate the pose of the bot after resampling a few times.

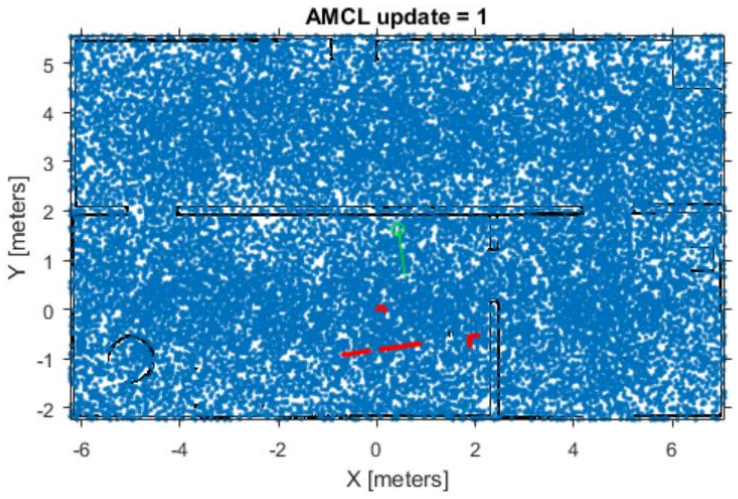

Fig 7 – Estimated pose of bot after the first update

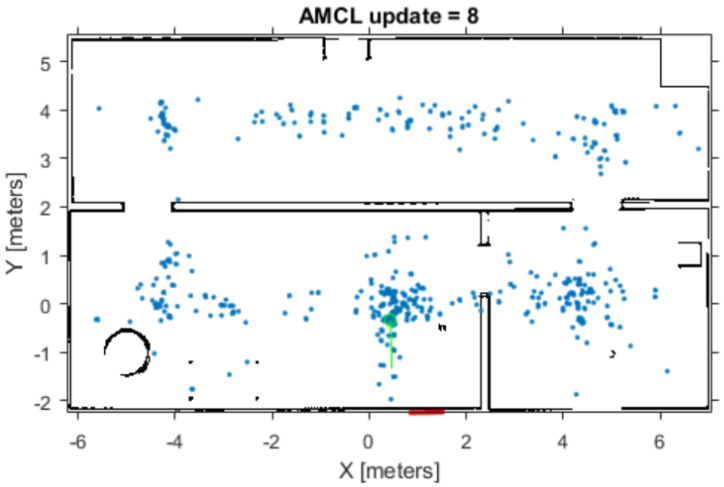

Fig 8 – Estimated pose of the bot after 8 updates

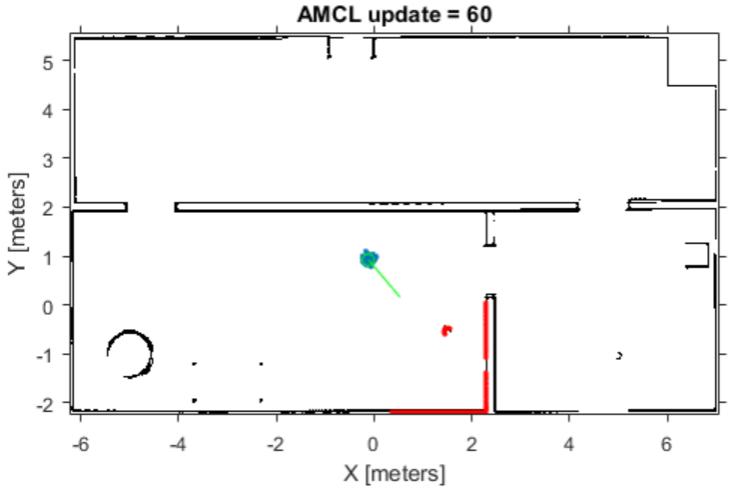

Fig 9 – Estimated pose of the bot after 60 updates

The values of d and φ are given by the wheel encoders. Using these we can calculate the new pose [xn, yn, θn] from the old pose [x, y, θ]

$$xn = x + d * cos(\varphi + \theta) \quad ...(1)$$
$$yn = y + d * sin(\varphi + \theta) \quad ...(2)$$
$$\theta n = \varphi + \theta \quad ...(3)$$

The particle filter starts with a number of particles spread over the map at all possible values where the bot can be and converges on the true pose of the bot after a few updates. The filter corrects its probability after each update using the current odometric value along with the map & ultrasonic sensor readings, and resamples those values to give the estimated pose and covariance. The filter updates its state if the odometric pose changes more than a set threshold value. The convergence of particles for a particular global localization scenario is shown in fig 7, 8 & 9. As we can see in this scenario, the MCL filter gives the true pose of the bot over 60 updates.

4. Navigation Model

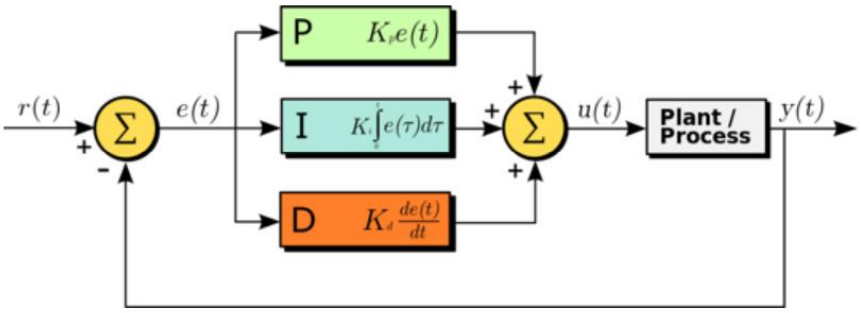

Fig 10 – PID controller

The controllers of this bot are based on PID control, which stands for “Proportional - Integral - Derivative”, whose control mechanism depends on the feedback and state (pose) information obtained from the MCL filter.

As we can see in the fig 10:

Term P depends on the current value of the error. For instance, if the error is small and negative, the control output will be in proportion, i.e., small and negative, taking into consideration a gain factor "K". Using P values alone, without I and D values – in this case changes to the input of the bot - will cause an error between the current bot position and the input value, because it requires an error to generate the proportional response. Without any input, there would not be any corrective response.

Term I considers all the past values of error, and then using integration, produces the I value. For instance, if error persists after application of proportional control, the integral term comes into action by integrating all the past errors, which accumulate over time. Then it gives an error correction output depending on the present value. In the way, this term has provision for all the errors committed during the past.

Term D is used to estimate future probable errors, depending on the rate of change obtained from differentiation. It is sometimes called "anticipatory control", as it reduces the SP-PV error by finding its current rate. As we know, differentiation gives the rate of change, so depending on the current rate, it predicts future errors, dramatically increasing the speed of the system.

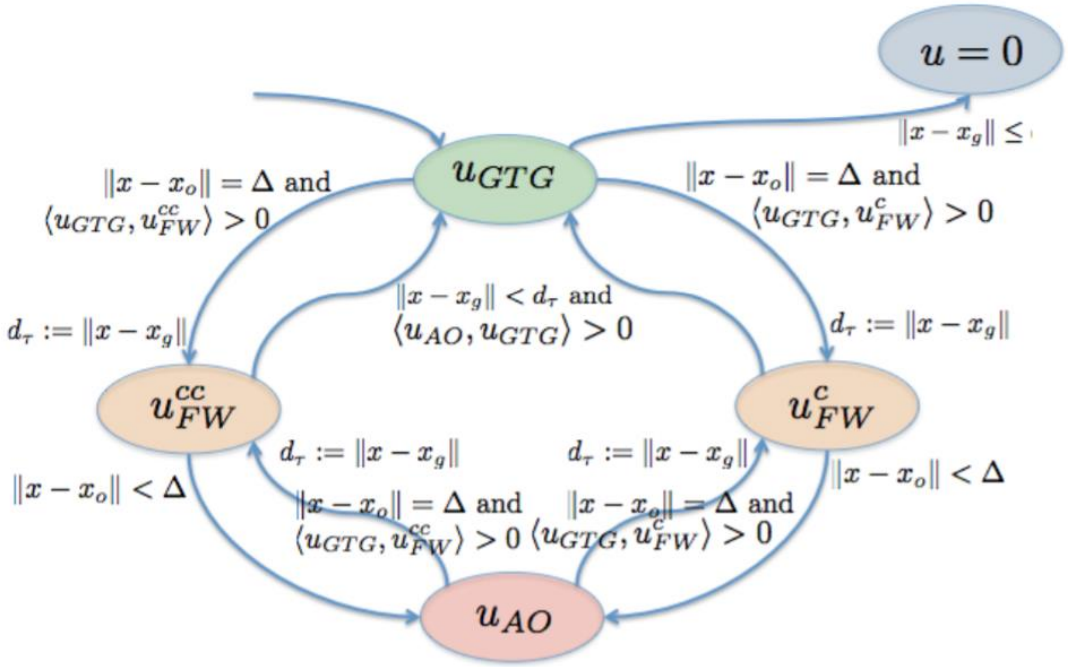

Fig 11 – Complete Navigation Model

Fig 11 shows the complete navigation model algorithm. It consists of 4 controllers in total to track the goal point while efficiently avoiding the obstacles encountered along the course. The arrow signifies the direction of change in flow of control from the current controller to the next, if the labelled conditions are satisfied. *x* is the current position of the bot [x, y] as returned by the MCL algorithm, x_o is the position of the obstacle as returned by the IR sensors and x_g is position of the goal to be tracked, given by the map. Once x_g is reached i.e. $x = x_g$, the x_g value is updated to the next target position specified in the map.

The four key controllers of the navigation model are:

a) ***go-to-goal*** **(u_{GTG}) –**

This controller is in effect when no obstacle is in proximity.

$$u_{GTG} = K_{GTG} * (x_g - x)$$

b) *avoid-obstacles* **(u_{AO}) –**

This controller is in effect when no obstacle is in proximity.

$$u_{AO} = K_{AO} * (x - x_o)$$

c) *follow-wall-clockwise* **(u_{FWC}) –**

This controller is in effect when no obstacle is in proximity.

$$u_{FWC} = \alpha R(-\pi/2) * u_{AO}$$

d) *follow-wall-counterclockwise* (u_{FWCC}) –

This controller is in effect when no obstacle is in proximity.

$$u_{FWCC} = \alpha R(+\pi/2) * u_{AO}$$

Here,

- $K_{GTG} = \frac{v_o\left(1-e^{-\alpha||e||^2}\right)}{||e||}$
- $K_{AO} = \frac{1}{||e||}\left(\frac{c}{||e||^2+\epsilon}\right)$
- $R(\theta) = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}$
- c, ϵ, α are gain constants & v_o is velocity limit

Now, v & ω for bot is obtained from the equation:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1/l \end{bmatrix} R(-\theta) \begin{bmatrix} u_x \\ u_y \end{bmatrix}$$

Here, θ is heading of the bot & l is the distance between the point for which the controllers are designed and the actual point of control on the bot. It should be small but not very small otherwise bot starts spinning.

v_R & v_L calculated as follows:

$$v_R = \frac{2v + \omega L}{2R}$$

$$v_L = \frac{2v - \omega L}{2R}$$

Here, L is distance between propellers & R = $\frac{Pitch\ of\ Propeller}{2\pi}$

Now, rpms of each propeller are related to v_R & v_L as:

$$RPM_R = \frac{v_R(mph) \times 1056 \times Gear\ Ratio}{Pitch(in)}$$

$$RPM_L = \frac{v_L(mph) \times 1056 \times Gear\ Ratio}{Pitch(in)}$$

5. Deploy for Cleaning

After the map is computed, we can deploy the bot for cleaning at any time from the MATLAB app by pressing the respective button as shown in fig 1. On deploying the bot for cleaning, the bot first localizes itself using the Monte Carlo Localization algorithm, while the bot localizes itself, it moves along the boundary of the map using the same follow wall algorithm used while mapping.

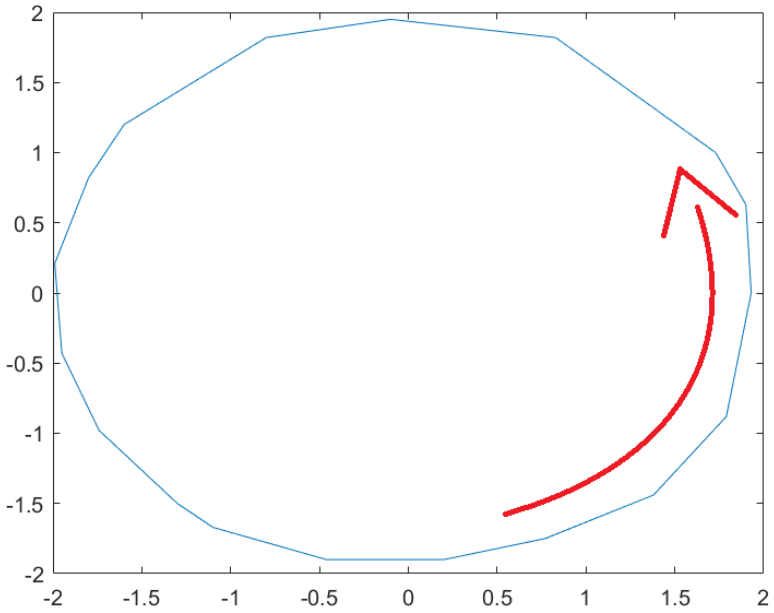

Fig 12 – Motion of bot while it localizes itself on deployment for cleaning

Once the bot localizes itself and a good estimate of the pose of the bot within the map is obtained, the navigation model kicks in. The navigation model works in conjunction with the Monte Carlo Localization algorithm now. The MCL algorithm continuously provides the navigation model with the current pose of the bot. The first goal point is provided by the map as shown in fig 5. The bot will go the first goal, regardless of wherever it is once the initial localization is done. Once one goal point is reached, the next one is given to the navigation model. If an obstacle is detected by the sensors when the bot is moving the towards the goal, the avoid obstacles or follow obstacle wall controllers are activated accordingly. In this way the bot is successfully able to track the goal points and move around the bounded region in a zig-zag pattern, effectively collecting the floating waste.

6. Circuit Diagram

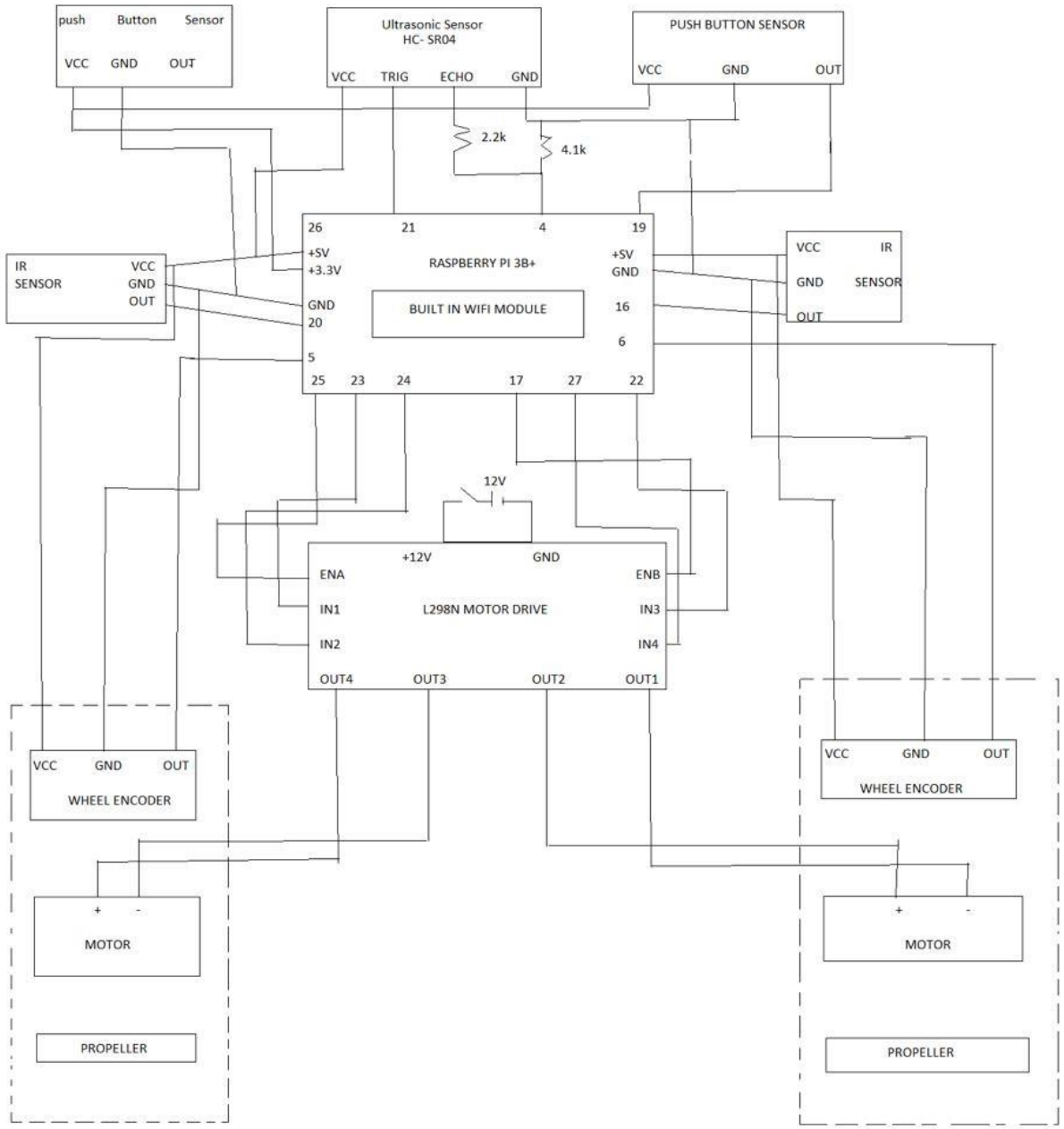

Fig 13 – Circuit diagram used for fabrication of the bot

7. Fabrication

The components used are as follows:

- Raspberry Pi 3B+
- HC-SR04 Ultrasonic Distance sensor
- Proximity IR sensors x2
- Push Button sensors x2
- 1.5V AA batteries x8
- AA battery holders x2
- L298N Motor Driver
- Float Level Control Switch
- Dual Shaft DC Gear Motors x2
- Wheel Encoder Disks x2
- IR sensors for wheel encoder disks x2
- RC Boat Propeller x2
- 32GB microSD card
- 5000 MAh Power Bank
- 2.1A Micro-USB Type B charging cable
- Breadboard
- Foam Board
- Wires
- Jumper cables
- Resistors

After making all the connections as shown in the circuit diagram, plug in the Raspberry Pi to the power supply to startup the bot. Once the Pi starts up, it is connected to the host PC where we run the MATLAB application code and perform the required tasks. The 12V power supply for the motors is provided by the AA batteries. We use a current divider circuit for connecting the ECHO pin of the ultrasonic sensor as the Pi is rated for 3.3V input voltage and anything more than that can damage the GPIO pins.

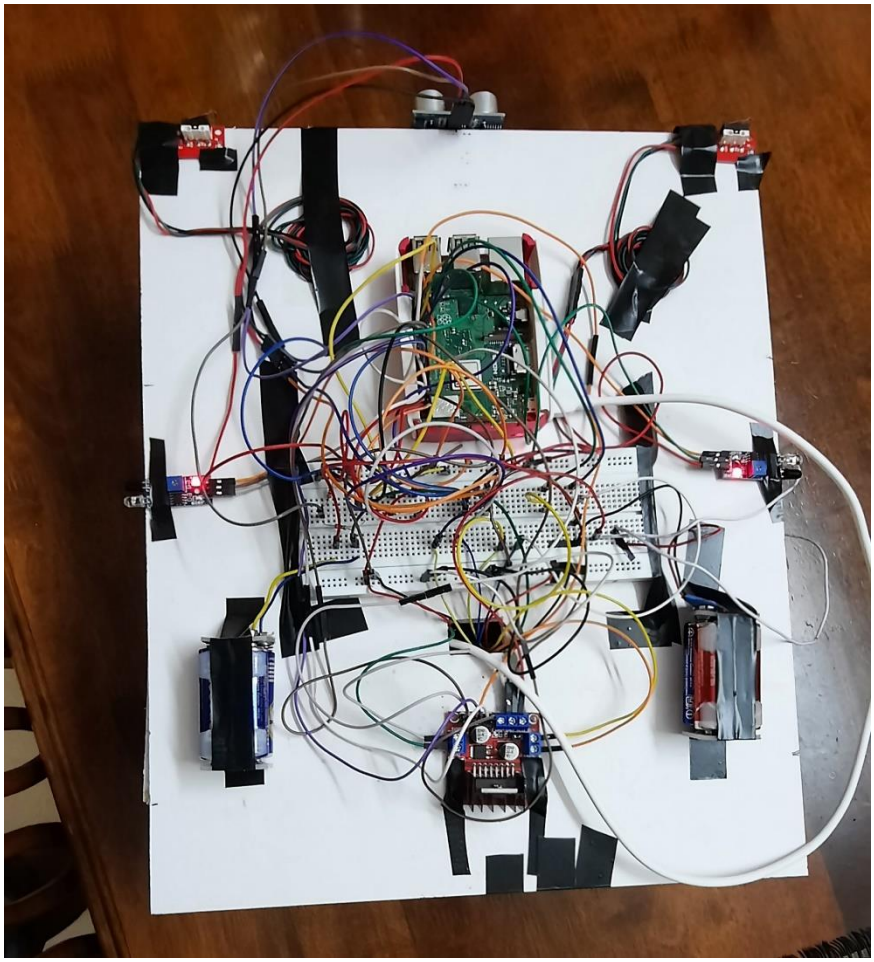

Fig 14 – Top View

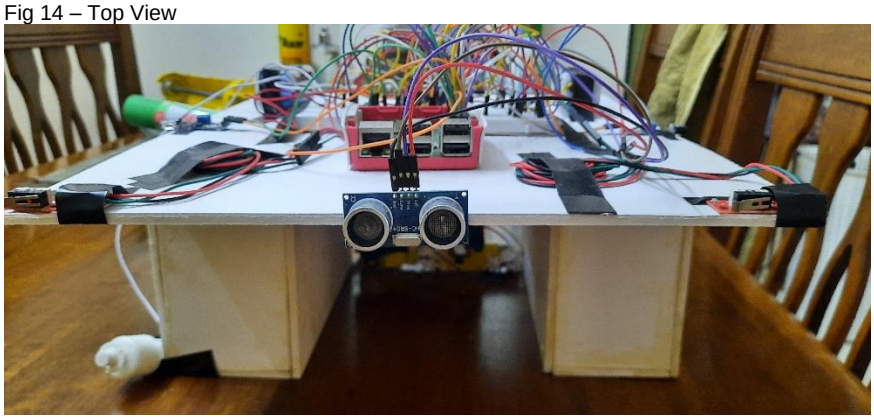

Fig 15 – Front View

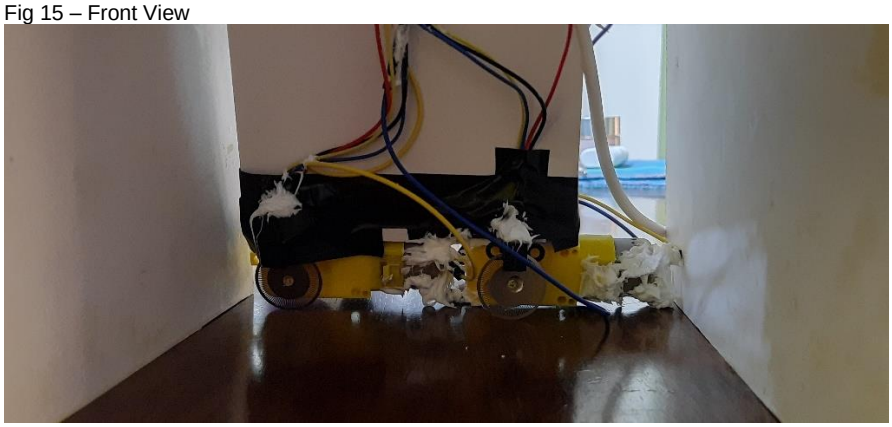

Fig 16 – Wheel encoders used for odometry

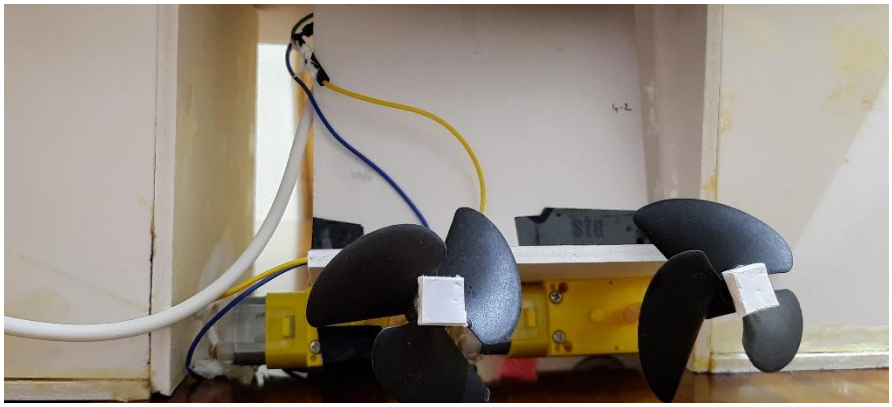

Fig 17 – Propellers

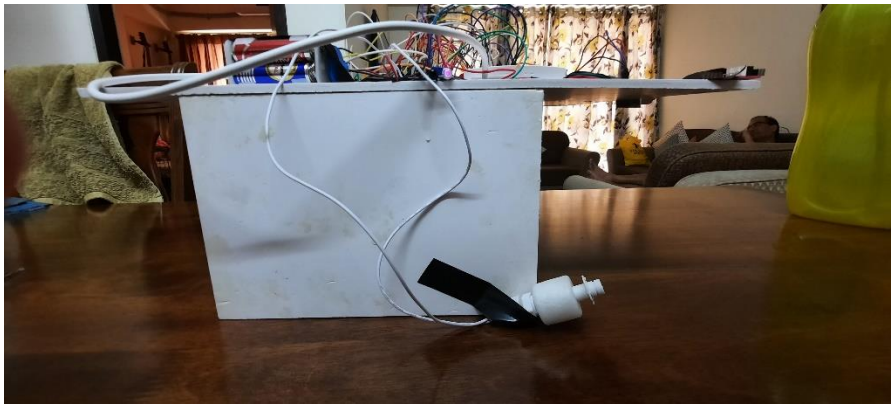

Fig 18 – Side View, tapped at the bottom is the float switch which gets activated in water

Fig 19 – 5000MAh power bank tapped on bottom side of the deck to power the Pi board

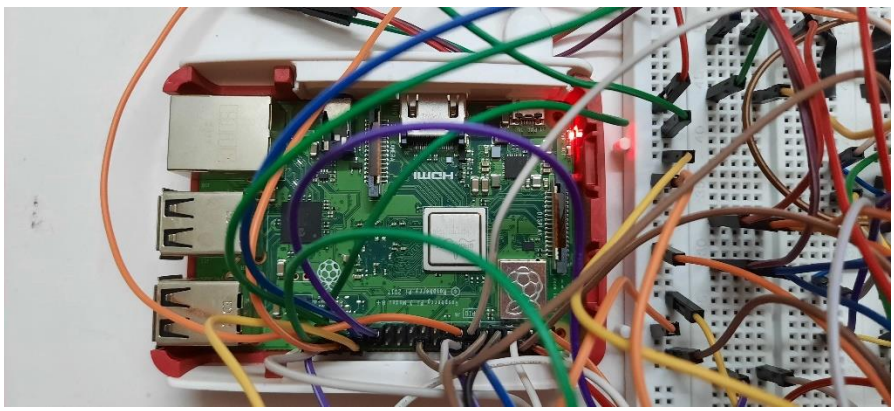

Fig 20 – Raspberry Pi connections

Fig 21 – Bot in action

8. Features

- It is a non-conventional river cleaning system.
- It's initial & maintenance cost is low.
- Skilled Worker not required to drive the system.
- Environment friendly system.
- Easy in operation.

9. Applications

- It is applicable to reduce water pollution in lakes & ponds.
- With little modification to the collection mechanism this bot can be used to clear the oil spillage on water bodies.
- It is applicable to reduce all types of impurities which are floating on water surface in swimming pool.
- It is useful to remove the environmental marine pollution at water reservoirs.
- It is useful in fishery plant to collect dead fishes.

10. Conclusion

We have successfully designed and fabricated an autonomous water surface cleaning bot which can be used for collecting floating waste of water surfaces, laying the foundation for future works in this direction.

11. Future Scope

The AWSC bot we have designed is suited to collect floating waste in small & still water bodies and would not work efficiently in large flowing waters like oceans and rivers. Future work on the bot can involve changing the design, incorporating stronger build and much more powerful motors & propellers to overcome the currents as well as making the respective changes in the software of the bot to account for upstream and downstream flow. For larger water bodies it would be sensible to use remote navigation controls rather than giving the bot complete autonomy. Also, different localization and mapping techniques would have to be used as we can no longer follow the boundary of such unbounded water bodies. Accordingly, changes would have to be made to the choice of sensors as IR and ultrasonic sensors would no longer be feasible for such a large-scale project, sensors like LIDAR, camera, GPS and IMUs would be better suited.

12. References

[1] D. Mukherjee, A. Saha, P. Mendapara, D. Wu, Q.M.J. Wu, “A Cost Effective Probabilistic Approach To Localization And Mapping,” IEEE Conference Paper, July, 2009.

[2] I. M. Rekleitis: “A Particle Filter Tutorial For Mobile Robot Localization”, Technical Report, TR-CIM-04-02, McGill University.

[3] I.J. Cox: ”Blanche - An Experiment In Guidance And Navigation Of An Autonomous Robot Vehicle”, IEEE Transactions on Robotics and Automation, Vol. 7, No. 2, pages: 193-204 (1991).

[4] "Introduction: PID Controller Design". University of Michigan.

[5] “Control of Mobile Robotics”. Gorgia Institute of Technology.

[6] R. Abiyev, D. Ibrahim , B. Erin, “Navigation of mobile robots in the presence of obstacles”, Near East University, Department of Computer Engineering, Mersin 10, Turkey.