

A custom-built robot wheelchair is shown from a top-down perspective. It features a central black battery pack with the handwritten text "IIT/CVIT/ASV1/2016-17/2" on its top surface. The battery is connected to a complex network of wires in various colors (red, black, green, blue, yellow) that lead to a central control unit and two DC motors. The motors are mounted on a metal frame and drive two large, treaded wheels. A breadboard with electronic components is visible on the right side of the battery. The entire assembly is mounted on a black frame, and the wheels are positioned on a light-colored tiled floor.

Robot Wheelchair

Ansh Khandelwal (2019102008)
Ashirwad Sinha (2019102035)
Charchit Gupta (2019102034)



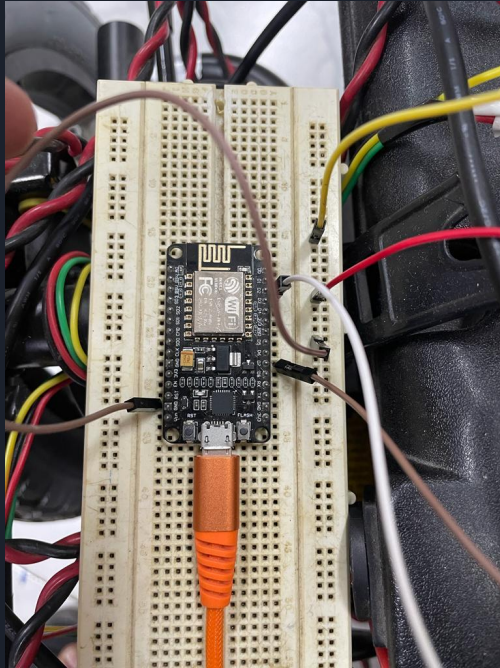
Introduction

Project divided into three parts:

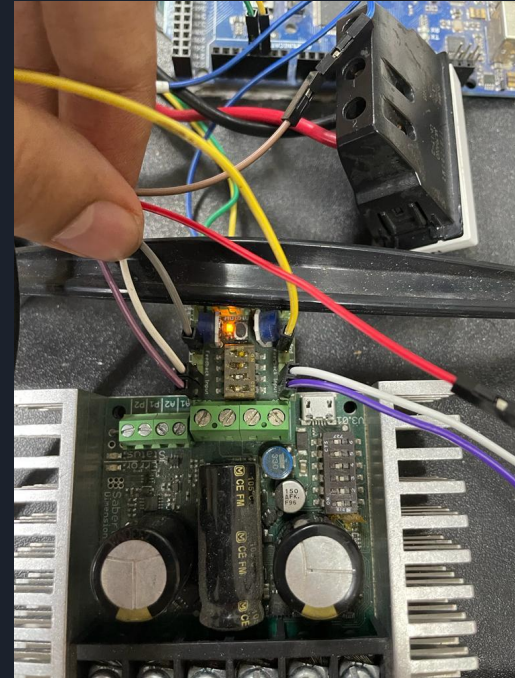
1. RPM and PPS calculation using the wheelchair encoder and an interrupt detector circuit implemented via arduino
2. Odometry calculation of the wheelchair using differential drive model
3. Pose graph optimization using 2D SLAM

Circuits and mode of operation

Interrupt detector circuit



Connections to sabertooth



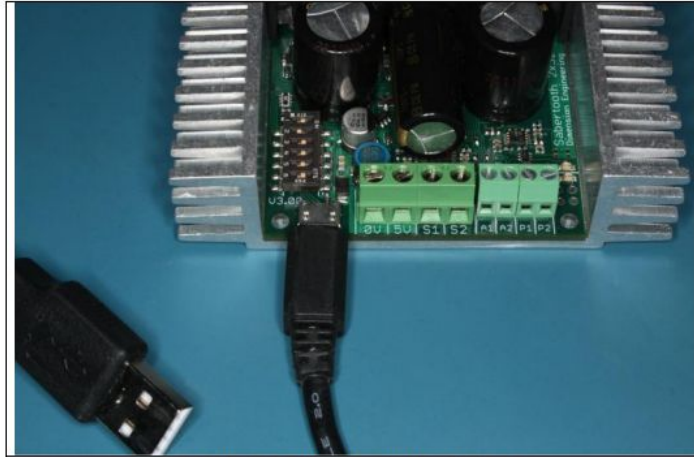
SABERTOOTH

Sabertooth 2x32 is a dual channel motor driver capable of supplying 32 amps to two motors, with peak currents up to 64 amps per motor. It can be operated from radio control, analog, TTL serial or USB inputs.

There are four control modes :-

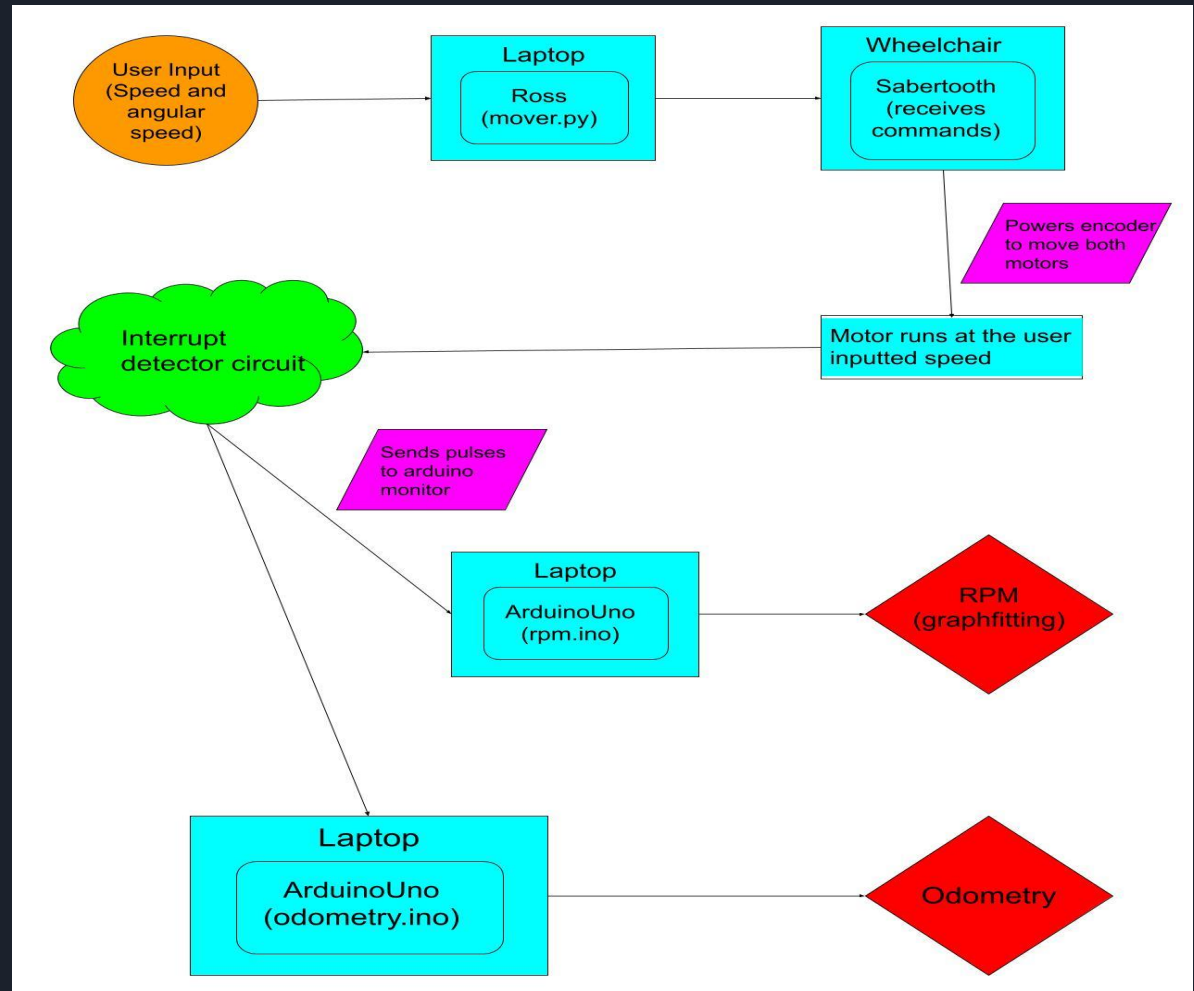
- Analog
- Radio control
- Serial
- USB mode


For our project we worked in the USB mode.



A Sabertooth 2x32 connected to a USB cable

Workflow of the project





RPM calculation: Why was this needed?

Initially the ros server was publishing the values into the server to control the speed of the motor. However, the speed of the motor was unknown, there was no unit of calculation.

What have we found?

From regressive experimentation, we have plot the values of the speed of motor with respect to input ros values and made a graph function. This function gives direct relation between the input parameters and the actual speed metrics.



RPM: How did we calculate it?

We first calculated the number of pulses per second (pps) through interrupt signal .

From pps we calculated rpm by multiplying it with 60 and dividing by gear ratio and pulses per revolution (ppr) .

$$\text{RPM} = (\text{pps} * 60) / (\text{gear_ratio} * \text{ppr})$$

Gear_ratio : It is the ratio is the ratio of the number of rotations of a driver gear to the number of rotations of a driven gear .

Pulses per revolution (or PPR): It is a parameter associated with encoders. Basically, it is a measure of the number of pulses per full revolution or turn of the encoder, with a full revolution being 360 degrees

Code snippet and working

- We define the interrupt pin as D2 and set baud rate 9600.
- Now after every interrupt we are incrementing cnt variable to count the number of pulses.
- In order to average we are taking a window of 5 seconds to calculate the rpm of the wheel.
- After that rpm is calculated by the formula:
$$\text{RPM} = (\text{pps} * 60) / (\text{gear_ratio} * \text{ppr})$$

```
const int interruptPin = D2;
double cnt=0,t1=0,t2=0,flag=0,fcnt=0,pps;

void setup() {
  Serial.begin(9600);
  pinMode(interruptPin, INPUT);
  attachInterrupt(digitalPinToInterrupt(interruptPin), IntCallback,RISING);
}

void loop() {

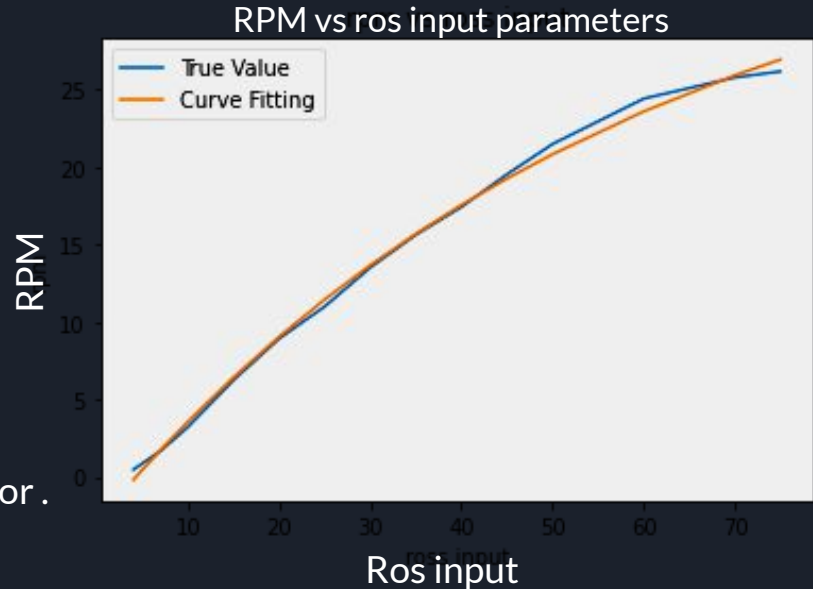
  if(flag ==1)
  {
    pps = fcnt/5;
    rpm_current = 0.00036621 * pps;
    Serial.println(rpm_current);
    flag =0;
  }
}

ICACHE_RAM_ATTR void IntCallback(){
  t2=millis();
  if(t2-t1>=5000){
    t1=t2;
    fcnt= cnt;
    cnt=0;
    flag = 1;

  }
  cnt++;
}
```


Result: Graph Fitting

- From our experiments we came to conclusion that rpm was not a linear function of ros input.
- It is rather is the form : $y = a - b \cdot \exp(-cx)$
 $a = 38.4035349$ $b = 41.3227983$ $c = 0.0170967158$
where $y = \text{rpm}$ and $x = \text{ros input}$.
- We tried linear and square root functions also but the exponential function gave us the least mean square error.



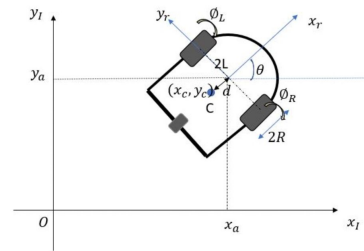
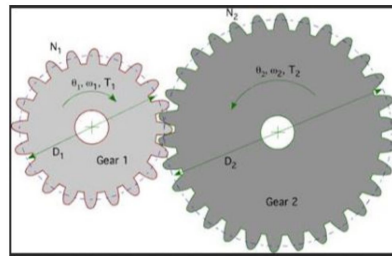


Odometry

Odometry is the use of data from the motion sensors to estimate the change in position over time. This method is sensitive to errors due to integration of velocity measurements (ex: RPM) over time to give position estimates.

In our project we used the **differential drive model** to estimate the trajectory of the wheelchair.

Deriving the kinematic/differential drive model



Gear Box (left) , Differential Drive Robot

- Kinematic constraints:
 - No slipping occurs.
 - Lateral velocity in robot's frame = 0 => $\dot{y}_r = 0$

$$-\dot{x}_a \sin(\theta) + \dot{y}_a \cos(\theta) = 0$$
- Let $\dot{\phi}_{R1}, \dot{\phi}_{R2}, \dot{\phi}_{L1}, \dot{\phi}_{L2}$ be the angular velocities of the corresponding wheels. Then,

$$v_R = \dot{\phi}_{R2} R$$

$$v_L = \dot{\phi}_{L2} R$$

$$v_R = \dot{x}_a \cos(\theta) + \dot{y}_a \sin(\theta) + L\dot{\theta} = \dot{\phi}_{R2} R$$

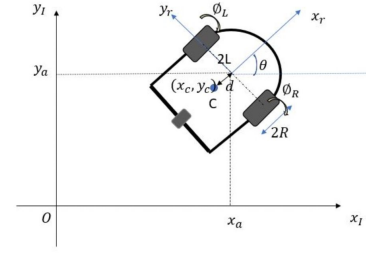
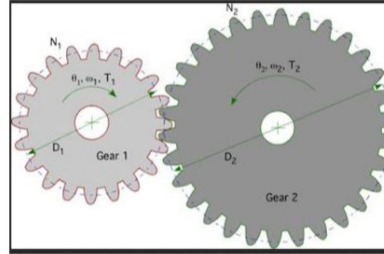
$$v_L = \dot{x}_a \cos(\theta) + \dot{y}_a \sin(\theta) - L\dot{\theta} = \dot{\phi}_{L2} R$$
- The gear ratio, $N = N_2/N_1$, where N_2 and N_1 be the number of teeth in the corresponding wheels, then:

$$\dot{\phi}_{R1} = -N\dot{\phi}_{R2}$$

$$\dot{\phi}_{L1} = -N\dot{\phi}_{L2}$$
- Now let there be a vector q defined as:

$$q = [x_a \quad y_a \quad \theta \quad \phi_{R1} \quad \phi_{R2} \quad \phi_{L1} \quad \phi_{L2}]^T$$

Deriving the kinematic/differential drive model



Gear Box (left) , Differential Drive Robot

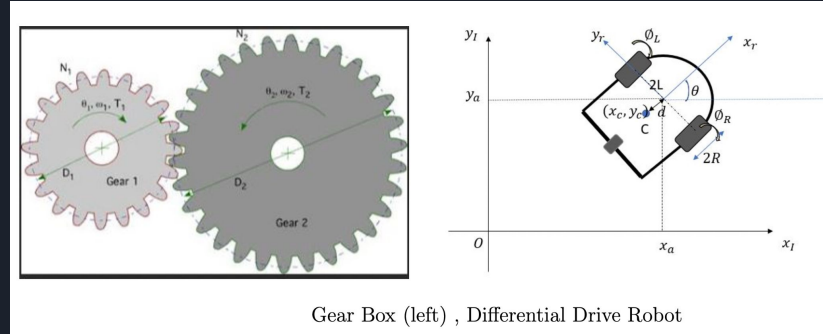
- Solving all the equations we arrive at:

$$\begin{bmatrix} -\sin(\theta) & \cos(\theta) & 0 & 0 & 0 & 0 & 0 \\ \cos(\theta) & \sin(\theta) & L & 0 & -R & 0 & 0 \\ \cos(\theta) & \sin(\theta) & -L & 0 & 0 & 0 & -R \\ 0 & 0 & 0 & 1 & N & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & N \end{bmatrix} \dot{q} = 0$$

- Moreover the governing kinematic equations of the module can be calculated by taking the angular velocities as inputs as follows:

$$\begin{bmatrix} \dot{x}_a \\ \dot{y}_a \\ \dot{\theta} \\ \dot{\phi}_{R1} \\ \dot{\phi}_{R2} \\ \dot{\phi}_{L1} \\ \dot{\phi}_{L2} \end{bmatrix} = \begin{bmatrix} \frac{-R}{2N} \cos(\theta) & \frac{-R}{2N} \cos(\theta) \\ \frac{-R}{2N} \sin(\theta) & \frac{-R}{2N} \sin(\theta) \\ \frac{-R}{2NL} & \frac{R}{2NL} \\ 1 & 0 \\ -\frac{1}{N} & 0 \\ 0 & 1 \\ 0 & -\frac{1}{N} \end{bmatrix} \begin{bmatrix} \dot{\phi}_{R1} \\ \dot{\phi}_{L1} \end{bmatrix}$$

Deriving the kinematic/differential drive model



- For calculating the odometry we only need the first three values of the output matrix. From Taylor series expansion we can approximate the following:
- In this way we can calculate the odometry values.

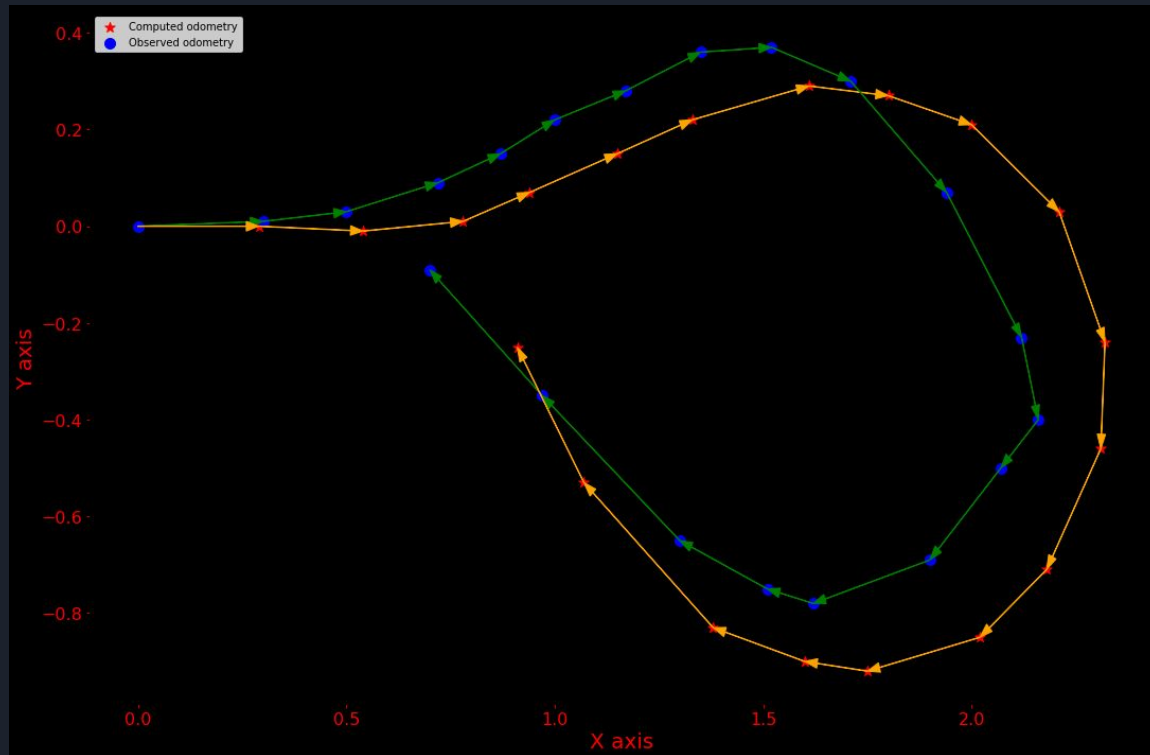
$$\begin{aligned}
 x(t + dt) &= x(t) + R \frac{\Delta\varphi_R + \Delta\varphi_L}{2} \cos \theta(t) \\
 y(t + dt) &= y(t) + R \frac{\Delta\varphi_R + \Delta\varphi_L}{2} \sin \theta(t) \\
 \theta &= R \frac{\varphi_R - \varphi_L}{2b}
 \end{aligned}$$

should depend on the previous

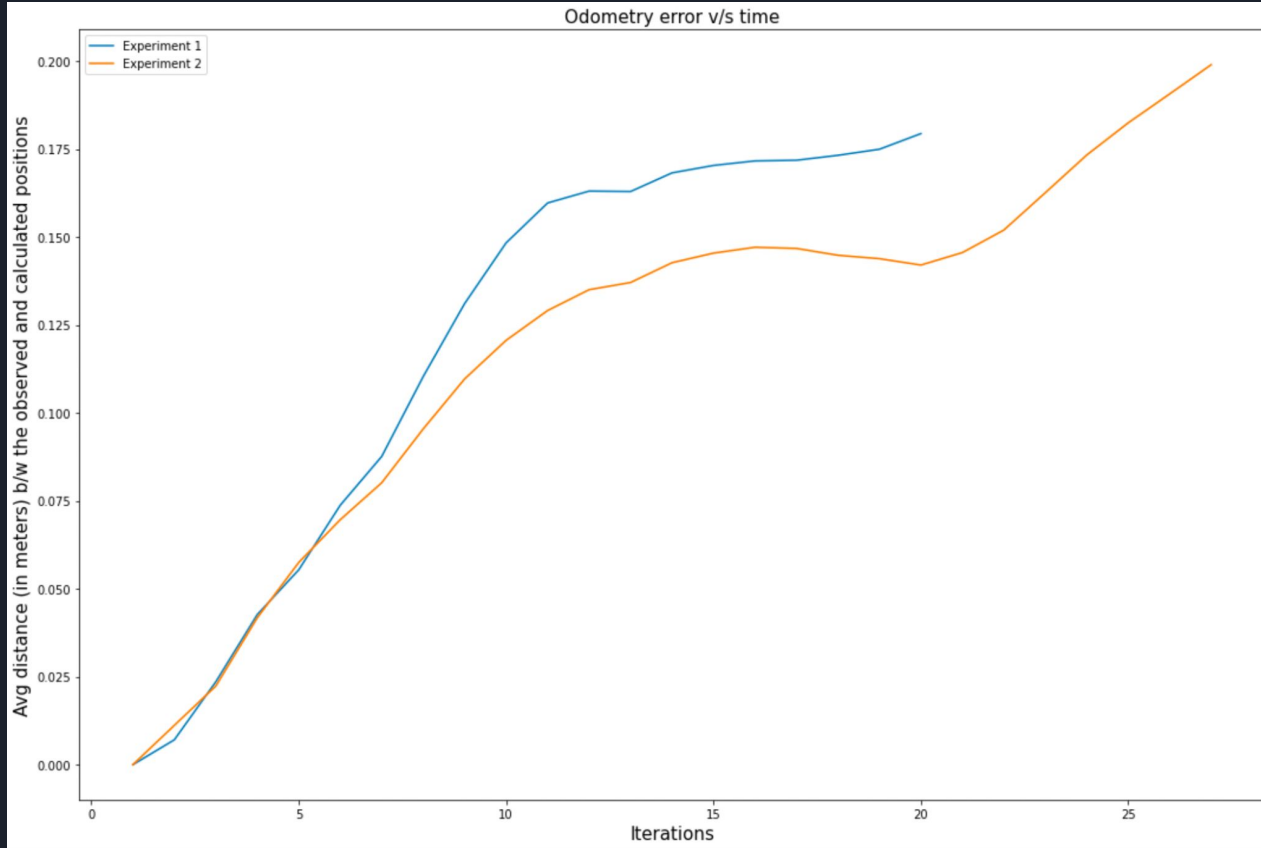
Results: Odometry

- Video demonstration link:
https://drive.google.com/file/d/1mWFqCd9qfCXwZUT_2Vg2EjH2xXPYtPof/view?usp=sharing

Odometry



Odometer deteriorates with time



Potential reasons for inaccurate odometry

1. Wheel Slipping.
2. Interrupt preference.
3. High iteration gap time in the differential drive model.

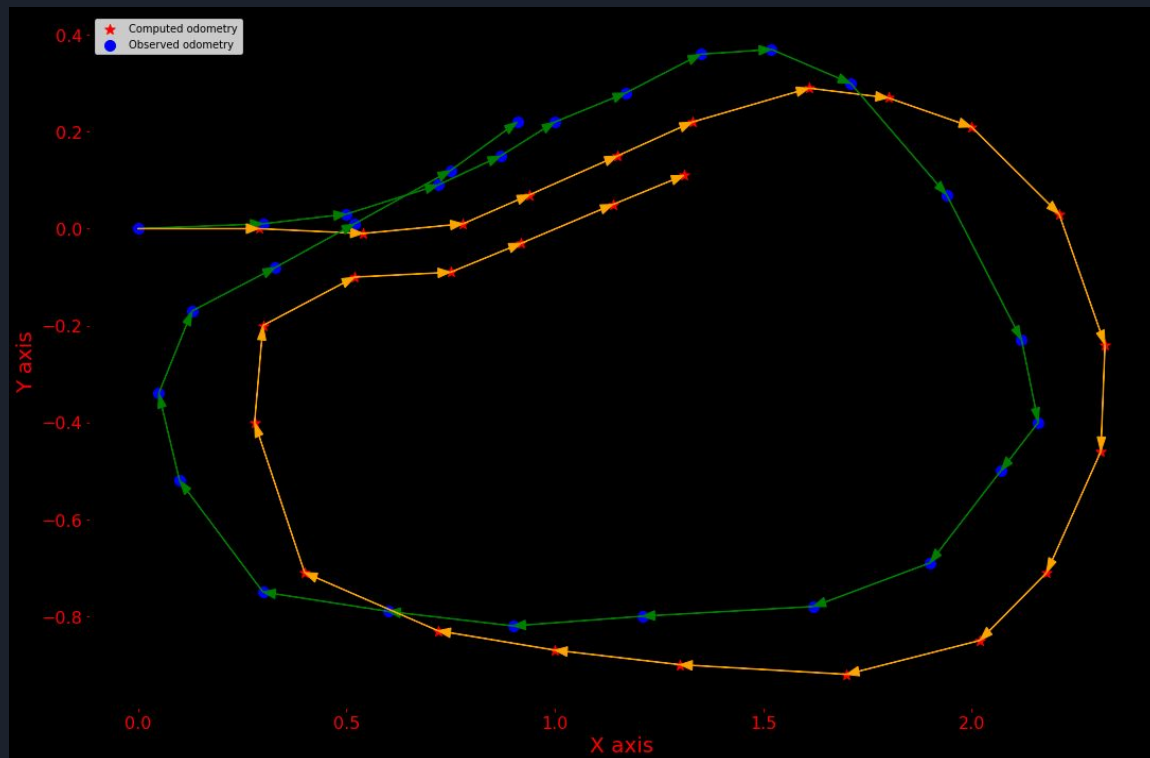
Can we improve the odometry?

Solution: Pose graph optimization using **LOOP CLOSURES**.
Loop closing is the task of deciding whether or not a vehicle has, after travelling arbitrary amount of length, returned to a previously visited area.

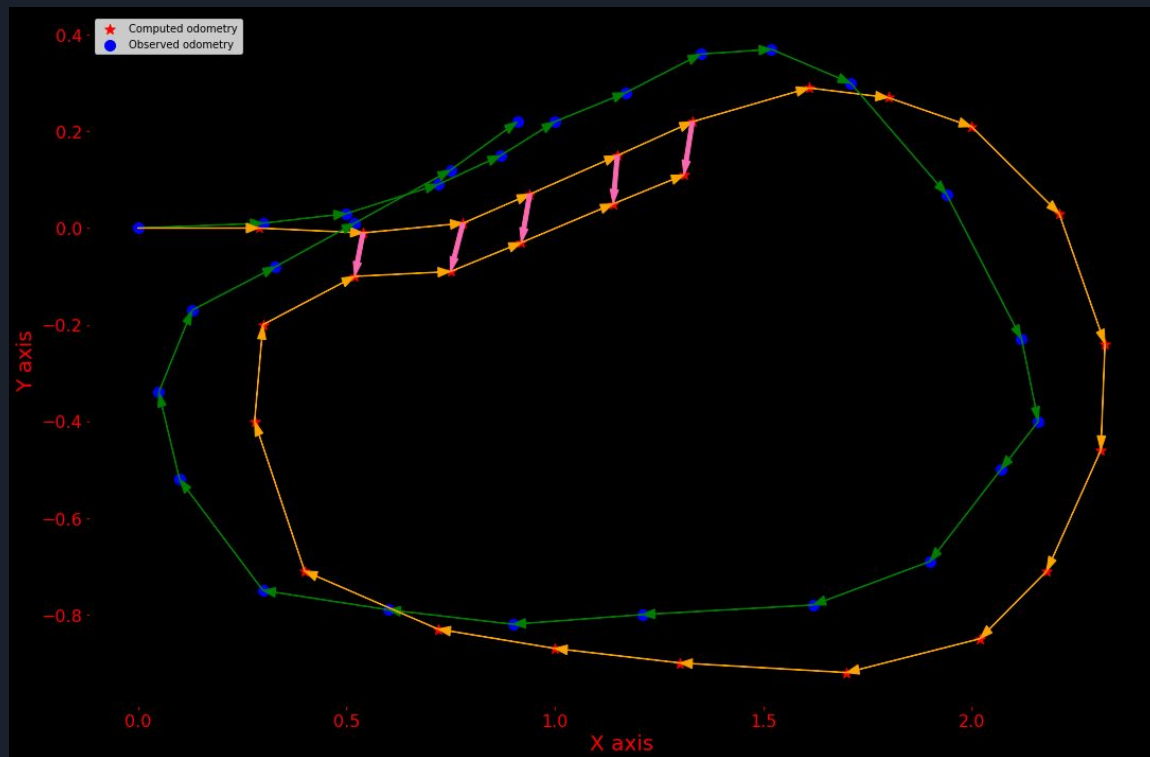
Even very small error gets accumulated over time and the actual trajectory looks no way near the calculated one, loop closures tries to take the calculated odometry close to the observed one.

Let's look at an example from our own wheelchair experiment.

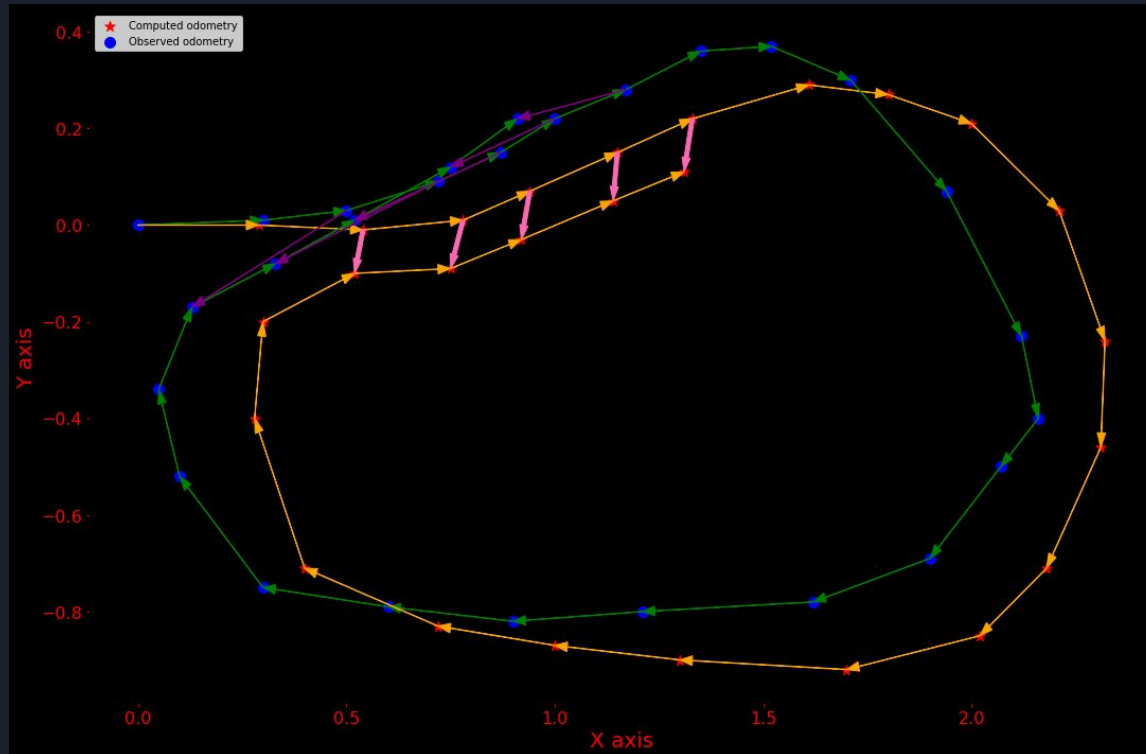
Loop Closures



Loop Closures



Loop Closures



Loop closure edges can be found by tracing a set of **Landmarks** from the wheelchair.

2D SLAM

Pose graph optimization

Consider the following variables:

Anchor: position 0

Odometry information = 100

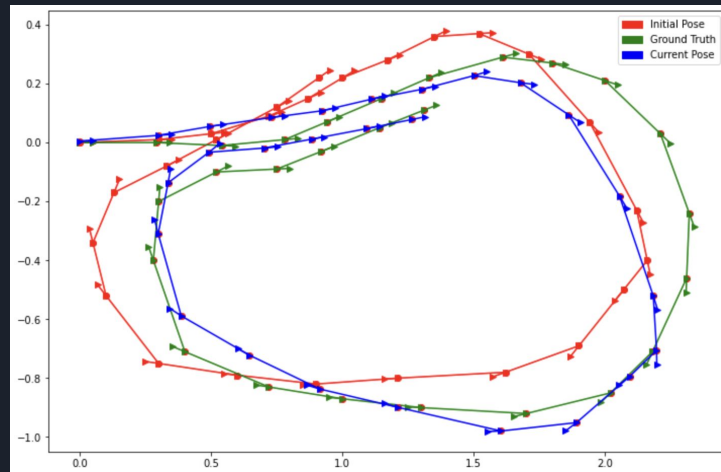
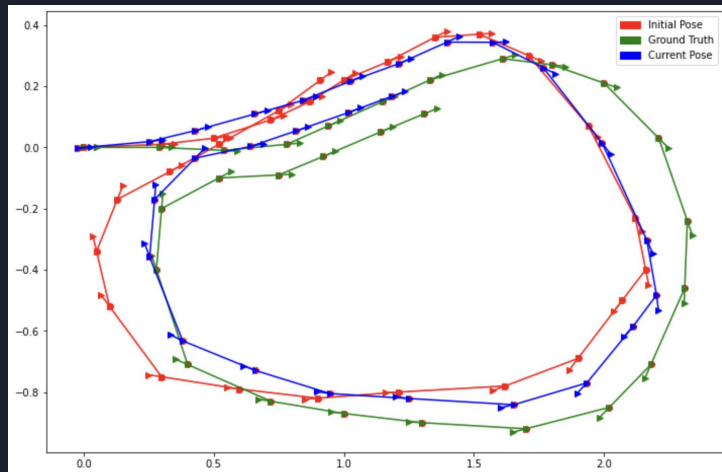
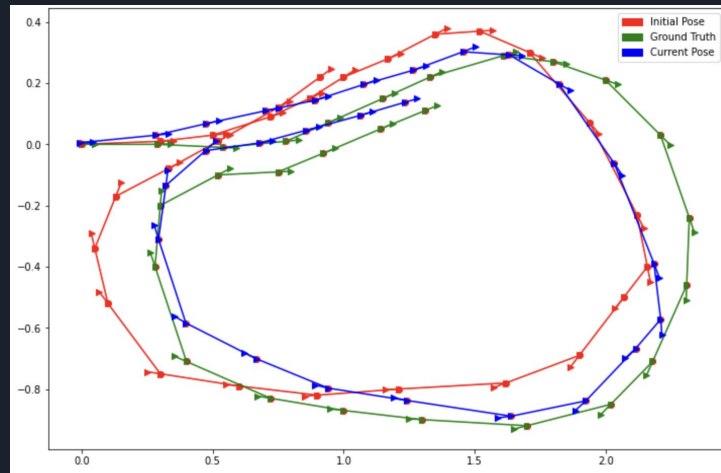
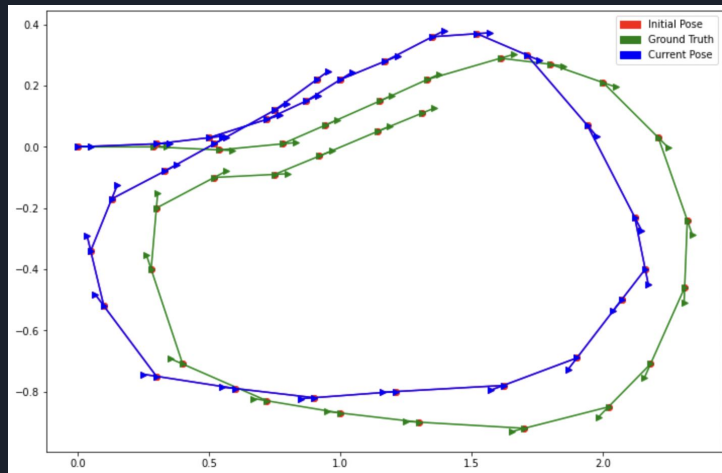
Loop Closure information = 1000

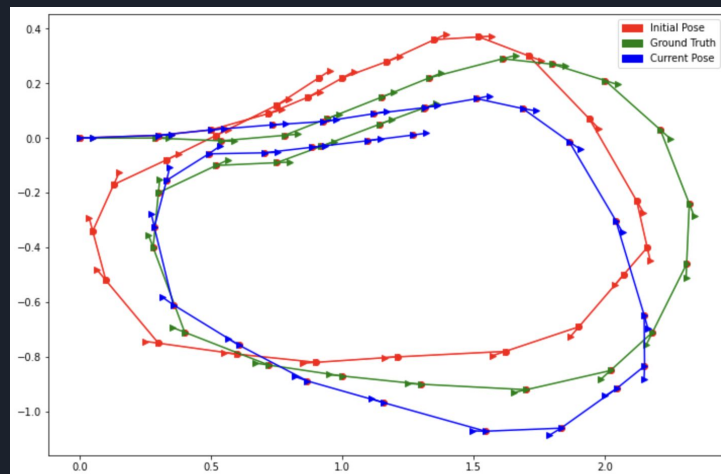
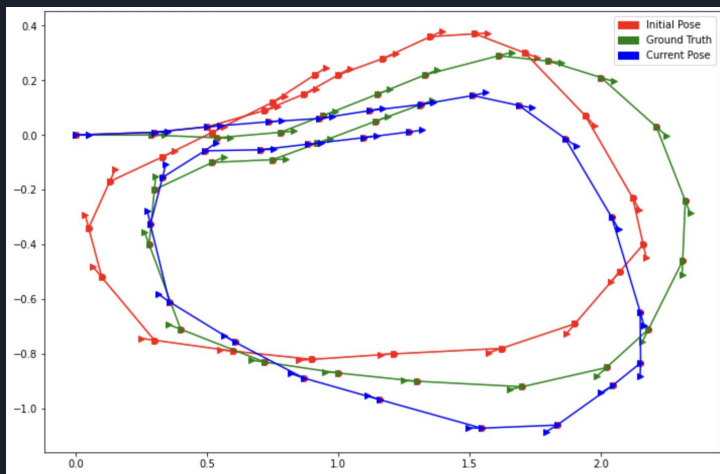
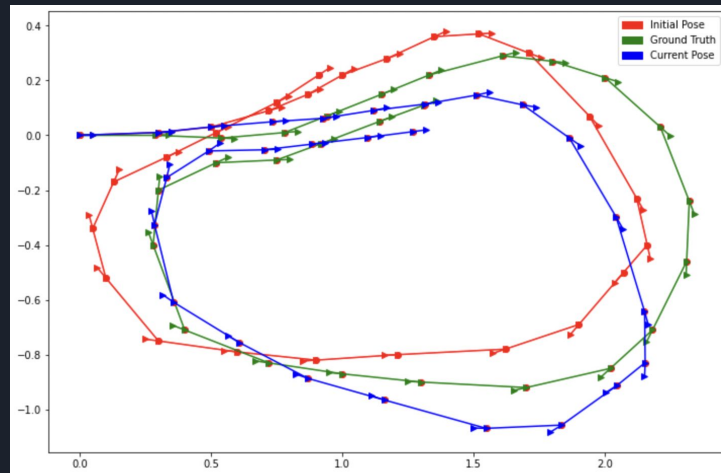
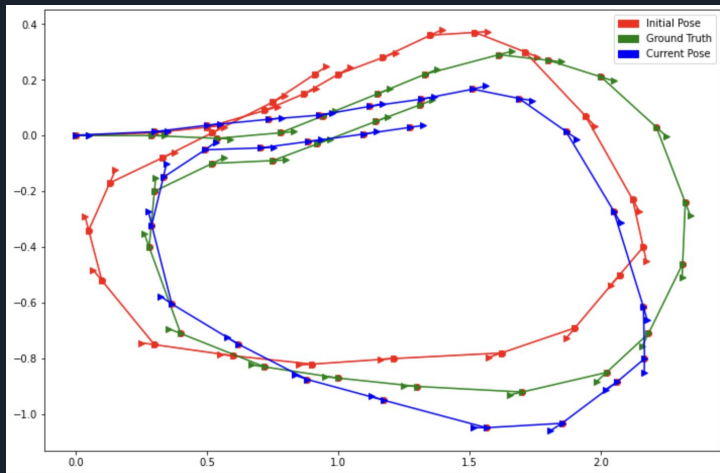
Anchor information = 100

The information values indicate how sure we are of them.

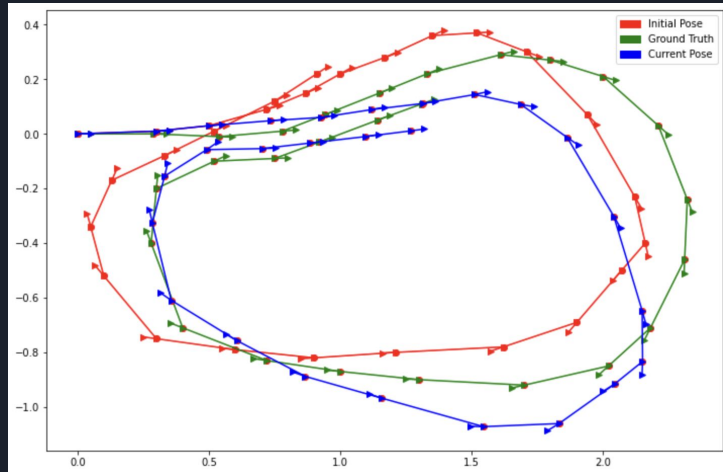
Now, **let's run the 2D SLAM** on our calculated odometry.

Note: Here we are **using the ground truth odometry to calculate the loop closure edges**. In reality we would not have the ground truth and would need to find loop closure using RGBD sensors to spot landmarks.





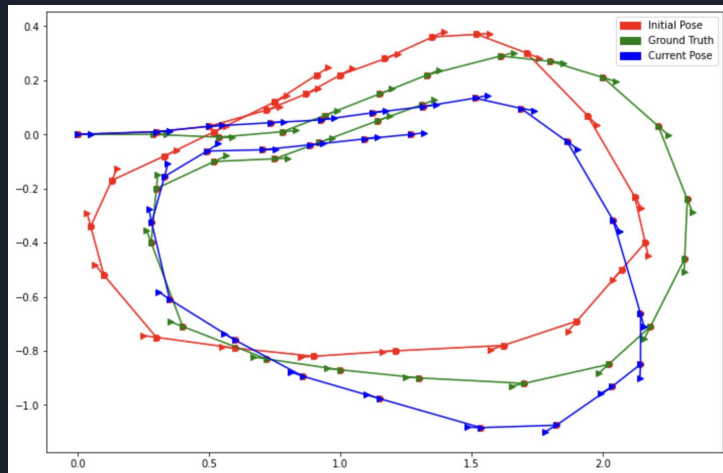
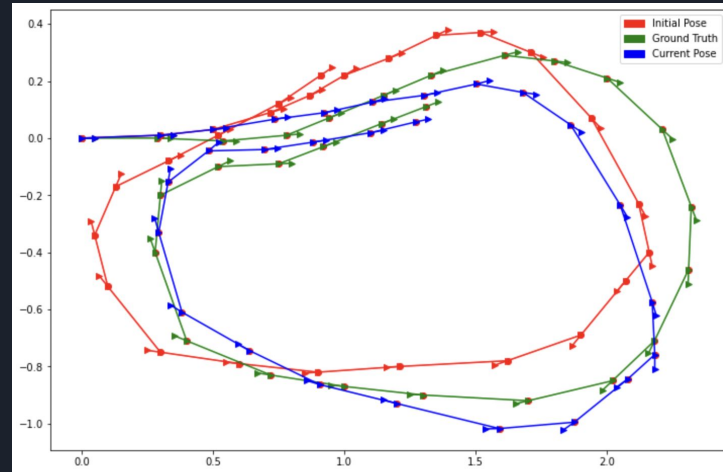
Let's try varying the information parameters!



Odometry Loop Closure Anchor

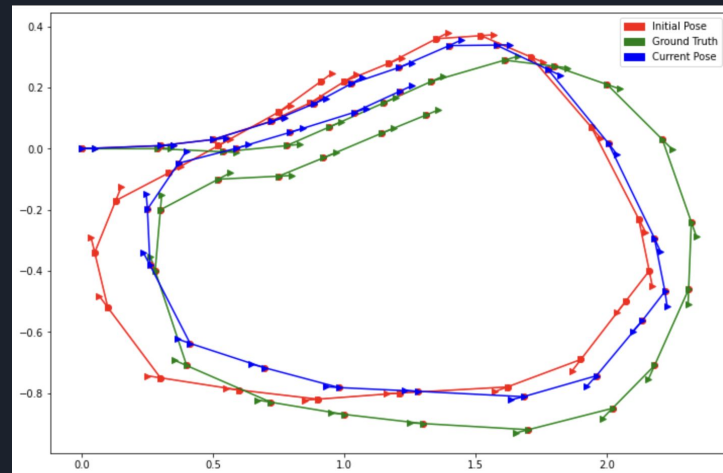
100
1000
100

500
700
1000



1
100
100

600
20
10



Best Results

After running from different information parameters we found the best ones to be:
Information odometry = 200
i 1000, 5000)

