ENMT482 Robotics

Assignment 1 – Group 19

Student

Zoren James Dela Cruz	98672643	zjd15@uclive.ac.nz
Ellis Lawrence	82612826	ewl17@uclive.ac.nz

2020

University of Canterbury

1.0 Sensor Fusion

In Part A, an Extended Kalman Filter (EKF) was designed to estimate the position of the robot. To implement sensor fusion, the group selected sonar 1, 2 and infrared sensor (IR) 3 to estimate the position of the robot as they had the smallest variance. Fusing these sensors, reliable sensor readings could be recorded between 0-5m which was more than required for this project.

1.1 Sensor Models

Deriving the sensor models for sonar 1, 2 and IR 3, the sensor outliers were removed for the 'calibration.csv' dataset. The methodology used to remove outliers was the Iglewicz and Hoaglin modified Z-score as, shown in Equation 1. The limitation of this method was that it needs a sensor model to remove outliers; however, this was not the case for this assignment. Given the situation, this equation was modified to use the median of a set of sensor readings to determine if any value in the set was an outlier.

$$M_i = 0.6745 \frac{(\epsilon_i - median(\epsilon))}{median(|\epsilon|)} \tag{1}$$

After removing the outliers from the sensor data, the sensor was then modelled using the gradient descent method. The gradient descent method was selected as the outliers were not entirely removed. The gradient descent was preferred over linear least squares as this method would not be as heavily affected by outliers. The derivation for the objective function for gradient descent in this project is shown in Appendix 1.1. Also, the sensor models and sensor data with outliers removed are displayed in Figure 4 from Appendix 1.2.

From the sensor readings, the variance increases as distance increases. Modelling the variance for each sensor with respect to distance was derived by removing the sensor outliers with respect to the sensor model and then applying a rolling variance. A polynomial fit function was applied on the output for the rolling variance to determine the coefficients for the variance model. The variance model for each sensor with respect to distance is shown in Appendix 1.3.

1.2 Sensor Fusion

As mentioned, sonar 1, 2 and IR 3 were chosen as their sensor models contained the least noise and covered the required range. The three sensors were fused using best linear unbiased estimate (BLUE), as shown in Equation 2, as the group's estimation approach to calculate estimated distance and its variance based on the sensor readings.

$$\hat{Z}_{BLU} = \frac{\sum_{i=1}^{N} \frac{1}{Var[Z_i]} Z_i}{\sum_{i=1}^{N} \frac{1}{Var[Z_i]}} \qquad Var\left[\hat{Z}_{BLU}\right] = \frac{1}{\sum_{i=1}^{N} \frac{1}{Var[Z_i]}} \tag{2}$$

The derivation to calculate the estimated distance for each sensor, Z_i is shown in Appendix 1.4. For IR 3, the calculation for Z_i was needed to be linearized as IR 3 is a non-linear model. Also, in the sensor fusion code, the group ensured that the sensors would only be fused if the robot's position was in the range of the sensor reading. The code also checks if the sensor readings are feasible by comparing its converted voltage to distance reading with the previously estimated position of the robot. This ensures that an outlier from the sensor reading would not be fused with other sensor readings that may affect the estimated position of the robot.

1.3 Motion Model

In this project, the motion model designed by the group devised a way of accounting for slip experienced by the robot and the time it takes to accelerate to its command speed. The modelled speed is illustrated by a piecewise function as shown in Equation 3.

$$Speed = \begin{cases} (0.77 + 0.005t_i)u_n & u_n > 0 \\ (0.77 - 0.005t_i)u_n & u_n < 0 \\ u_n & u_n = 0 \end{cases} \tag{3}$$

Where u_n, is the command speed and t_i is a constant that tracks if the previous and current command velocity are equal. If the condition is true, t_i increments by 0.01. The group's designed speed motion model tries to replicate the robot accelerating to its command speed overtime. After modelling the speed, the displacement was then calculated using Equation 4.

$$Distance = Previous\ Distance + \ Speed * dt \tag{4}$$

The motion model's effectiveness was illustrated in Appendix 1.5 through Figure 6.

1.4 Extended Kalman Filter

To predict the robot's position, an EKF was used to merge the sensor measurements and prediction. An EKF is a recursive algorithm which only needs to maintain the posterior belief and its variance. EKF was used for this project due to the non-linearity of the IR 3 sensor model.

1.5 Results

Applying the Kalman Filter, the robot's position and variance for 'test.csv' and 'training1.csv' is shown in Figure 1.

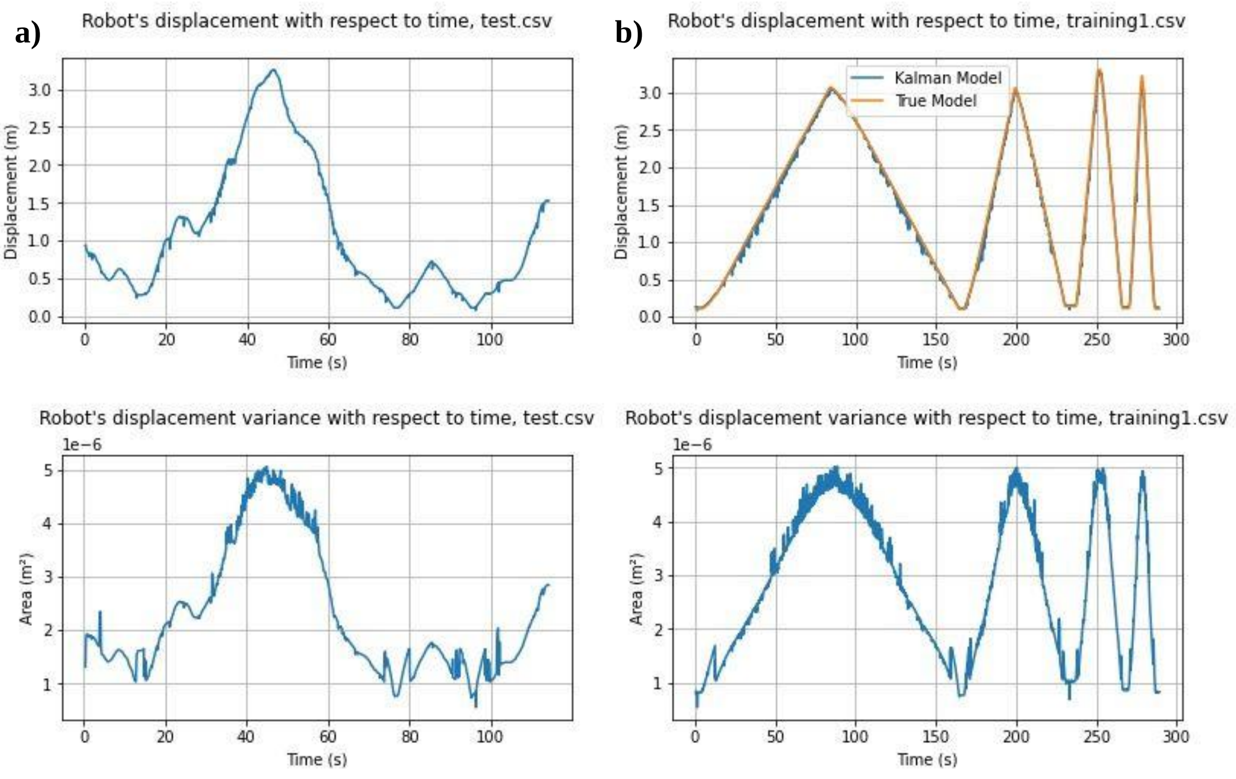


Figure 1: a) test.csv b) training1.csv

1.5 Discussion
In this project, the sensor model could have been improved by using a different algorithm to remove sensor outliers. This would have enabled the group to use a simpler parametric identification such a linear least square to estimate the sensor coefficients. On the other hand, the sensor fusion code developed has several strengths. The sensor fusion code ensured that the sensor readings that would be fused are within its range specified and those sensor readings outputted by the sensors were not outliers. This allowed the group's sensor fusion model to be effective as no erroneous readings would have affected the measured position of the robot which would have been merged with the predicted position of the robot in the EKF.

The motion model designed by the group showed improvements in comparison to using command velocity only. As shown in Figure 6, using the modelled speed tracks the true displacement better compared to using the command speed. The speed motion model shown in Equation 3 replicates an equation of a line ($y = mx + c$). The constant, c, for the speed motion model accounts for the slip the robot experiences. The linear constant, m, accounts for the robot accelerating to the command speed. This motion model designed can be improved by systematically calculating the coefficients used for the speed motion model rather than iteratively tuning. Also, reviewing the motion model code, if the current and previous velocity command do not change over time, the modelled speed will exceed the command speed. To prevent this, the code will need to cap the modelled speed to not exceed command speed.

Overall, the EKF designed by the group managed to track the robot's position effectively. Seen from Figure 1, the true model and the EKF model were nearly identical. However, a reoccurring pattern was that the variance of posterior belief tends to follow the same shape as the displacement model. This is because the sensor variance increases when distance increase.

2.0 Particle Filter
2.1 Motion Models
The motion model used for the particle filter was the odometry motion model. The odometry sensors on the wheels estimate the distance each wheel travels. An EKF is then used to fuse the odometry readings with the gyroscope measurements to estimate local pose. This odometry motion model uses the change in local pose to estimate the change in global pose.

In the code, the group first converted from global pose to local pose for the odometry readings. The group then samples from a joint PDF, $f_{\boldsymbol{D}}(\boldsymbol{d}:\boldsymbol{d'})$ where its standard deviation was iteratively tuned to improve the motion model. The code then updates the global pose for the particle filters, as shown in Equation 5. The unknown constants shown in Equation 5 are derived and described in Appendix 2.1.

$$\begin{aligned} x_n^{(m)} &= x_{n-1}^{(m)} + (d + d_e)\cos\left(\theta_{n-1}^{(m)} + \phi_1 + \phi_{1_e}\right) \\ y_n^{(m)} &= y_{n-1}^{(m)} + (d + d_e)\sin\left(\theta_{n-1}^{(m)} + \phi_1 + \phi_{1_e}\right) \\ \theta_n^{(m)} &= \theta_{n-1}^{(m)} + \phi_1 + \phi_{1_e} + \phi_2 + \phi_{2_e} \end{aligned} \tag{5}$$

Where $x_n^{(m)}$, $y_n^{(m)}$ and $\theta_n^{(m)}$ are the estimated pose for the m^{th} particle; d, ϕ_1 and ϕ_2 are the displacement, initial turn and final turn as shown in Figure 7 in Appendix 2.1; d_e, ϕ_{1_e} and ϕ_{2_e} are the added random error for the m^{th} particles estimated pose calculated from the joint PDF.

2.2 Sensor Models

In the sensor model code, the range and bearing measured by the robot's camera to the closest beacon's position were calculated. The code then calculates the range and bearing measured by the *m^th^* particle to the closest beacon's position. The equations used to calculate the range and bearings are shown in Appendix 2.2. The group then calculates the weight of each particle using Equation 6.

$$a_n^{(m)} = a_{n-1}^{(m)} f_R\left(r_n - r_n^{(m)}\right) f_\Phi\left(angdiff\left(\Phi_n, \Phi_n^{(m)}\right)\right) \tag{6}$$

Where $a_n^{(m)}$ and $a_{n-1}^{(m)}$ are the current and previous weights of the particle, r_n and $r_n^{(m)}$ are the ranges measured by the robot and *m*th particle, Φ_n and $\Phi_n^{(m)}$ are the bearings measured by the robot and *m*th particle, $f_R(r_n)$ and $f_\Phi(\Phi_n,)$ are the range and bearing error PDF.

2.3 Results

Figure 2 compares the particle filter estimated path against the odometry readings and slam algorithm.

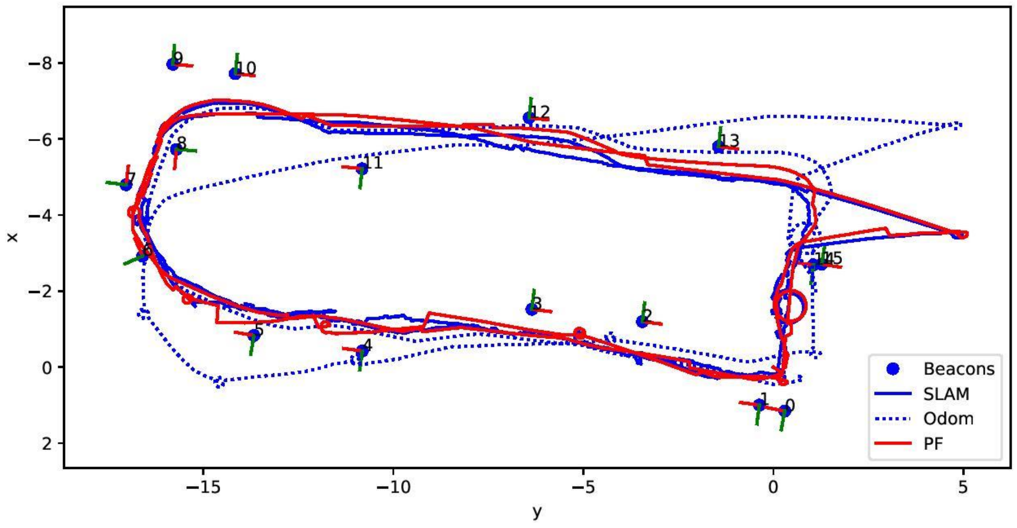


Figure 2: Mapping of the Particle Filter.

2.4 Analysis

The odometry motion model was used as it is more accurate than the velocity motion model. The velocity motion model does not take account of the possibility of the robot getting stuck. Whereas the odometry motion model keeps track of measurements. However, to further improve the motion model, the group could have combined the velocity and odometry motion model.

The sensor model to calculate the weight of the particles shown to be very effective during its run time. It shows that particles close to the SLAM track have higher weightings than the particles that drift away. And particles that have drifted far away are then resampled nearby the estimated pose.

Shown in Figure 2, the particle filter path tracks closely to the SLAM algorithm. There are occasions where the particle filter tends to drift away from the path but is then re-calibrated upon seeing a beacon as shown in coordinates (-8, -1.5) in Figure 2. This is because the particles with low weights would have been forced to reposition by the beacon.

An observation during demo.py run time, despite relatively good tracking of the particle filter, there are cases where the robot still gets lost and, a recovery method was needed to be performed. The particle filter is lost when the particle filter cloud was clustered in a small area with a low weighting rather than being spread. Due to this behaviour, the particle filter believes it is lost despite the particle filter path is travelling along the SLAM algorithm's track. The recovery method implemented by the group was to spread the particles uniformly around the previously estimated pose of the robot.

Another observation during demo.py run time was that robot tends to get caught between beacon 7 and 8. This could be because the chosen navigation beacon is continuously switching, and the robot is not certain which beacon to consider.

3.0 TurtleBot Mapping

The map shown in Figure 3 displays an aerial view of the automation lab. An occupancy grid was created with a 2D SLAM (simultaneous localisation and mapping) algorithm. This can construct a map while tracking the location of the TurtleBot. The shading seen is representative of the robot's certainty of whether a cell is occupied or unoccupied, represented by black or white shading respectively. The bold black lines are representative of the walls of the lab and neighbouring corridor, whereas the patches of localised black represent chair and desks within the room. Grey areas define map locations the robot has no data of.

The map produced from the TurtleBot was of reasonable local accuracy but, during the second loop of the lab, the TurtleBot lost its position which caused a duplication of the map. This likely occurred while the robot was moving down the outside corridor when it was stuttering or travelling too fast past the beacon positions. This inaccuracy demonstrates well why a self-scanner performs poorly in repetitive environments.

From this observation, as robot speed increases, the time taken for mapping decreases which cause the map quality to decrease. This is due to the sensor of the robot not having enough time to scan every area on the path it is travelling as computation must be completed before the next sensor readings are considered. The map update delay could also affect the quality of the final map; however, this was not accessible in the lab session.

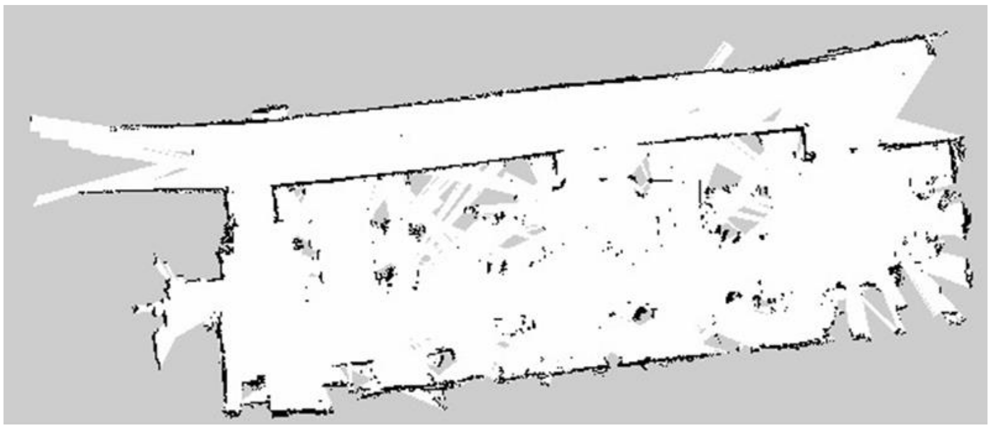


Figure 3: Automation lab mapping using gmapping algorithm.

1 Appendix – Part A

1.1 Part A - Gradient descent

Gradient descent is an iterative algorithm that determines the local gradient of the cost function and steps in the direction of steepest descent. This method minimises the cost function which enables the group to determine the sensor coefficients. Gradient descent is defined in Equation 7.

$$\boldsymbol{\theta}^{\boldsymbol{k+1}} = \boldsymbol{\theta}^{\boldsymbol{k}} - \alpha \nabla J(\theta^k) \tag{7}$$

Where $\boldsymbol{\theta}^{\boldsymbol{k+1}}$ and $\boldsymbol{\theta}^{\boldsymbol{k}}$ are the vector of the unknown parameters for the sensor coefficients, α is the learning rate and $\nabla J(\theta)$ is the slope of the cost function which is different for sonar and IR due to their models.

For Sonar sensors:

$$y = \theta_1 + \theta_2 x$$

$$J(\theta) = \sum_{i=1}^{n} (y_i - \theta_1 - \theta_2 x_i)^2$$

The slope of the cost functions is

$$\frac{\partial J(\theta)}{\partial \theta_1} = \sum_{i=1}^{n} -2(y_i - \theta_1 - \theta_2 x_i)$$

$$\frac{\partial J(\theta)}{\partial \theta_2} = \sum_{i=1}^{n} -2x_i(y_i - \theta_1 - \theta_2 x_i)$$

For Infrared Sensors:

$$y = \frac{\theta_1}{\theta_2 + x} + \theta_3 x$$

$$J(\theta) = \sum_{i=1}^{n} \left(y_i - \frac{\theta_1}{\theta_2 + x_i} - \theta_3 x_i\right)^2$$

The slope of the cost function is

$$\frac{\partial J(\theta)}{\partial \theta_1} = \sum_{i=1}^{n} -\frac{2}{\theta_2 + x_i}\left(y_i - \frac{\theta_1}{\theta_2 + x_i} - \theta_3 x_i\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_2} = \sum_{i=1}^{n} -\frac{2\theta_1}{(\theta_2 + x_i)^2}\left(y_i - \frac{\theta_1}{\theta_2 + x_i} - \theta_3 x_i\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_3} = \sum_{i=1}^{n} -2x_i\left(y_i - \frac{\theta_1}{\theta_2 + x_i} - \theta_3 x_i\right)$$

1.2 Part A – Sensor Models

In Figure 4, the sensor models after filtering are shown. The sensor outlier has relatively been removed by applying Iglewicz and Hoaglin Method to identify the outliers. With this method, no model for the sensor was provided; thus, the median from the set of data was used to remove the outliers within that dataset. Table 1 is the sensor model constants.

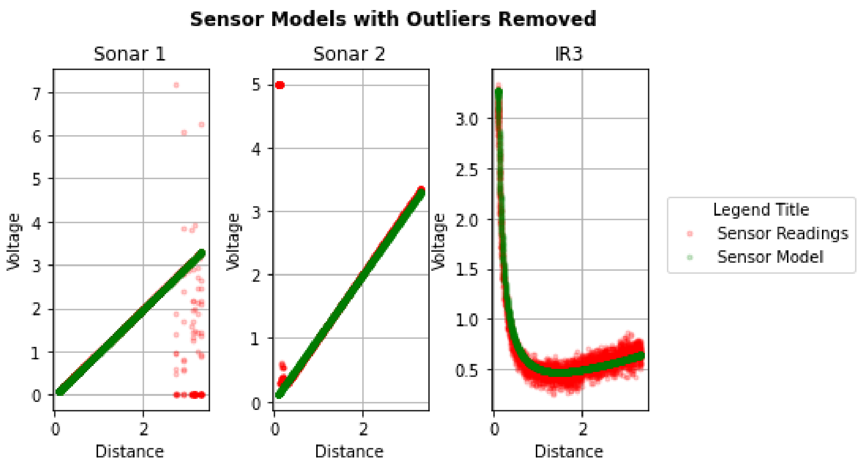


Figure 4: Sensor models, h(x)

Table 1: Sensor Model Constants

Sensor	Equation	Constants		
Sonar 1	$y = mx + c$	$m = 0.997$	$c = -0.03$	
Sonar 2	$y = mx + c$	$m = 0.982$	$c = 0.021$	
IR 3	$y = (k1/k2 + x) + k3x$	$k1 = 0.35$	$k2 = 0.01$	$k3 = 0.16$

1.3 Part A – Sensor Variance Models

In Figure 5, the sensor variance model and its sensor variance readings calculated from rolling variance are shown. The sensor variance model was calculated using the polyfit function to determine the variance polynomial coefficients. Table 2 is the variance model constants.

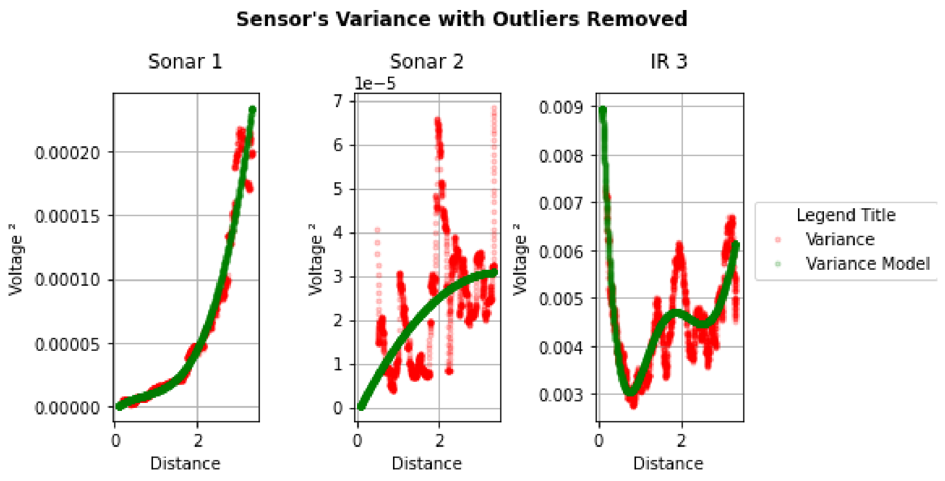


Figure 5: Variance Models, V(x)

Table 2: Variance Model Constants

Sensor	Polynomial Degree	a	b	c	d
Sonar 1	3	1.07×10^{-5}	-2.2×10^{-5}	2.42×10^{-5}	-1.1×10^{-6}
Sonar 2	2	-2.2×10^{-6}	1.87×10^{-5}	-1.4×10^{-6}	
IR 3	3	-3.9×10^{-4}	0.0025	−0.0042	0.0057

1.4 Part A – Convert Voltage to Distance

Converting the voltage readings to distance for Sonar:

$$Z_i = cX_i + D$$

Where c and D are constants calculated from sensor model, Z_i is the voltage reading and X_i is the measured distance. Rearranging the equation to calculate the measured distance from sensor reading.

$$X_i = \frac{Z_i - d}{c}$$

Converting the voltage readings to distance for IR:

$$Z_i = \frac{k_1}{k_2 + X_i} + k_3 X_i$$

To calculate the distance measured, the voltage readings need to be linearized.

$$\frac{\partial Z_i}{\partial X_i} = -\frac{k_1}{(k_2 + X_i)^2} + k_3$$

The linearized sensor reading for IR is

$$Z_i = \underbrace{\left[-\frac{k_1}{(k_2 + X_i^-)^2} + k_3\right]}_{h\prime(x_0)} (X_i - X_i^-) + \underbrace{\frac{k_1}{k_2 + X_i^-} + k_3 X_i^-}_{h(x_0)}$$

Where k_1, k_2 and k_3 are constants calculated from sensor model, X_i^- is the previous distance reading and X_i is the current measured distance. Rearrange the equation to obtain the current measured distance.

$$X_i = \frac{Z_i - h(x_0)}{h'(x_0)} + x_0$$

1.5 Part A – Motion Model

In Figure 6, the modelled speed described in section 1.3 was plotted against the command and estimated speed. Also, in Figure 6, the modelled speed was used to calculate the modelled distance which showed better tracking against the true distance.

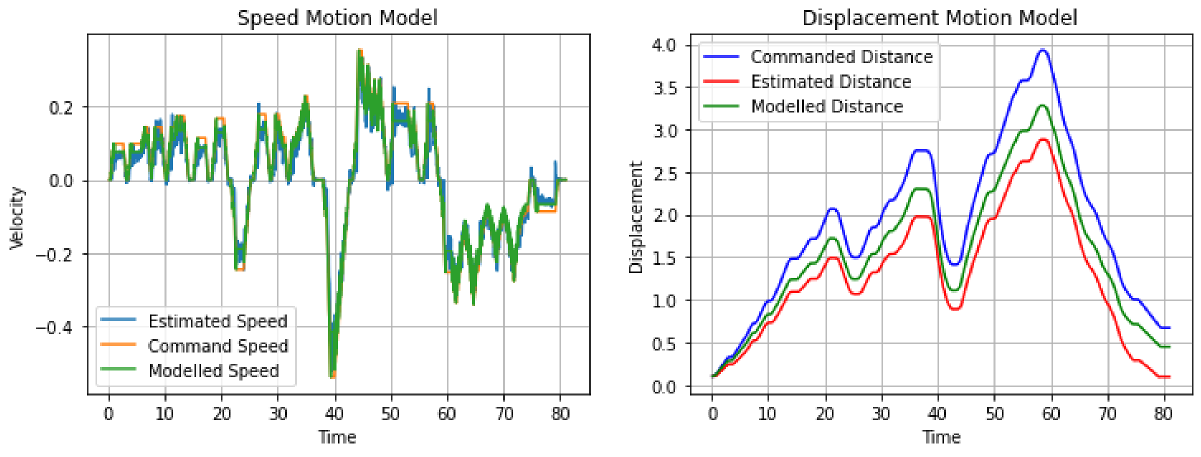
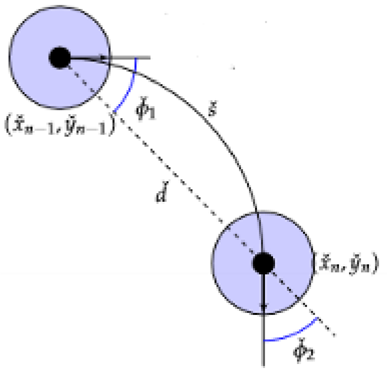


Figure 6: Motion Model graphs

2 Appendix – Part B

2.1 Motion Model

To convert the global pose to local pose for the odometry readings, the equations were derived from Figure 7.



$(\check{x}_{n-1}, \check{y}_{n-1})$
$\check{\phi}_1$
$\check{s}$
$\check{d}$
$(\check{x}_n, \check{y}_n)$
$\check{\phi}_2$

Figure 7: Derivation of motion model equations

$$d = \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}$$
$$\phi_1 = \tan^{-1}\left(\frac{y_n - y_{n-1}}{x_n - x_{n-1}}\right) - \theta_{n-1}$$
$$\phi_2 = \theta_n - \theta_{n-1} - \phi_1$$

Where x_n, y_n and θ_n were from the odometry readings, $d, \phi_1$and ϕ_2 are the displacement, initial turn and final turn.

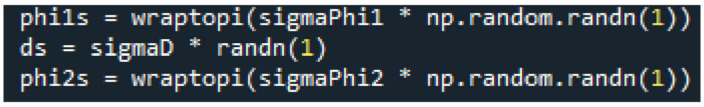
After converting to local pose, the standard deviation from the joint PDF was calculated for $f_{\boldsymbol{D}}(\boldsymbol{d}: \boldsymbol{d'})$ was calculated. The definition $f_{\boldsymbol{D}}(\boldsymbol{d}: \boldsymbol{d'})$ is shown below

$$f_{\boldsymbol{D}}(\boldsymbol{d}: \boldsymbol{d'}) = f_D(d: d')f_{\phi_1}(\phi_1, \phi_1')f_{\phi_2}(\phi_2, \phi_2')$$

The standard deviation for each of those joint PDFs was calculated using this equation.

$$\sigma_{\phi_1} = \alpha_1|\phi_1| + \alpha_2 d$$
$$\sigma_D = \alpha_3(|\phi_1| + |\phi_2|) + \alpha_4 d$$
$$\sigma_{\phi_2} = \alpha_1|\phi_2| + \alpha_2 d$$

Knowing their standard deviation, the sampled random error added to the particle poses was calculated as shown in the snippet of our code.



```
phi1s = wraptopi(sigmaPhi1 * np.random.randn(1))
ds = sigmaD * randn(1)
phi2s = wraptopi(sigmaPhi2 * np.random.randn(1))
```

2.2 Sensor Model

To calculate the range and bearing measured by the robot:

$$r = \sqrt{({}^r x_b)^2 + ({}^r y_b)^2}$$
$$\phi = \tan^{-1}\frac{{}^r y_b}{{}^r x_b}$$

To calculate the range and bearing measured by the *m*th particle:

$$r^{(m)} = \sqrt{(x_b - x^{(m)})^2 + (y_b - y^{(m)})^2}$$
$$\phi^m = angdiff\left(\tan^{-1}\left(\frac{y_b - y^{(m)}}{x_b - x^{(m)}}\right), \theta^{(m)}\right)$$

Where $angdiff(\theta_1, \theta_2)$ is a function that calculates difference between two angles so that its result is between $-\pi$ and π.