ROBO7001
Search and Rescue With Cozmo - Simulation, and Reality
Driving, Localisation, and Mapping

Authors:
Daniel Addo 19175148
Methuselah Singh 19188409
Sam Trowbridge 19182799

Word Count:
MSc Artificial Intelligence

Contents

1. Introduction

Robot localisation and mapping have the potential to dramatically impact future technology in the real world, however, successful implementation of this can be difficult. One solution proposed by Thrun et al. is a probabilistic approach. This project has been separated into two parts, the initial section included the implementation of a probabilistic sensor model and a mapping algorithm. This next segment aims to expand what has already been accomplished by applying kinematics with these two models. Followed by developing a probabilistic motion model and localisation model to improve motion and position accuracy. The final goal is to develop a Simultaneous Localisation and Mapping (SLAM) method, which combines each of the individual models into one complex algorithm.

After prior research, the first step is to design and implement an appropriate probabilistic motion model including kinematics and motion control. Experiments will be conducted to test the effectiveness by stating a target position for Cozmo to transverse towards. This followed with the development of exploration and navigation functionality, aiming to have Cozmo safely explore an environment and plot obstacles that may be present.

A proposed localisation algorithm will be designed, tested, and critically evaluated. These topics will culminate with the introduction, implementation, and evaluation of a SLAM algorithm for Cozmo to use within an unknown environment with distinct features. Features here refers to preprogrammed objects that it can recognise: the light cubes and wall label.

2. Motion Model and Driving Control

A Moton Model consists of an algorithm that verifies the probability of the current state given the previous state, velocity, and timestamp. Often a motion model will work in tandem with a systems Sensor Model (SM) to provide data to be implemented in localisation (Thrun et al., 1999).

The motion model algorithm below consists of three variables x_t: current position, x_{t-1}: previous position, and u_t: velocity, t represents the current timestamp.

$$p\left(x^t \mid u^t,\ x^{t-1}\right)$$

As Cozmo moves through an environment, various factors can affect its ability to correctly calculate its motion, leading to inaccurate results as will be discussed later in this report; the use of a motion model then allows it to accommodate this “noise” factors.

Motion models can be split into two different types. The first is an odometry model. These are predominantly implemented in autonomous systems equipped with wheel encoders. Specifically, the odometry model sends a motion command to the wheel encoder of the robot. This information is then used to estimate the distance a robot has moved, calculating from the starting position to either its current or target position (Thrun et al., 1999).

An alternative approach is a velocity-based model, however, these are implemented in systems without wheel encoders which means they do require a form of motion measurement beforehand. This motion model assumes control of a robot through a rotational, translational velocity and time elapsed.

When compared, odometry-based models give higher accuracy than velocity-based models; this is because the odometry model captures the rotations immediately after the command has been executed, while the velocity-based models have a high level of uncertainty whether a particular command has been executed or not**.** Cozmo can be implemented using both motion model types as it contains two magnetic encoders mounted on the shaft of the motors. These are the wheel encoders discussed above, which keep track of the exact motor speed.

Initial motion experiments

To better establish the baseline internal variances and errors of Cozmo, a variety of motion experiments were conducted. When conducting these experiments, effects due to external factors such as friction and drag which can accumulate, have not been taken into account. Also, due to the nature of shared use of university labs and robots, it was not entirely feasible or practical to be able to consistently use the same robot.

To determine how accurately it would stop at the correct distance, Cozmo was tested by driving a set distance in a straight line. Using motion equations to determine the wheel speed and time, values were programmed into the code using Cozmo's drive functionality.

Varying Track Speed and Time

Experiment	Track Speed	Time in 0.1s	Mean Actual Distance Travelled in mm	Range of Distance in mm
1	20	60	330.33	9
2	40	30	493.67	4
3	60	5	591.00	7

From this experiment, it can be determined that there is a level of inaccuracy, which has been observed through repeated tests, in Cozmo's motion, regardless of the task's simplicity of moving in a straight line.

Figure 1

Varying Track Speed only (time set at 60)

Experiment	Track Speed	Mean Actual Distance Travelled in mm	Range of Distance in mm
4	5	93.33	7
5	25	464.33	9
6	50	836.67	1
7	100	1674.00	10

These experiments provide evidence that given a set time and speed, there is still some inconsistency of distance traveled, demonstrated by the range from repeat experiments. However, from these results, it was noted that there is a minimum threshold for Cozmo's track speed, which if not taken into account, would mean that calculations for kinematics could result in a speed below this threshold; meaning the robot would no longer be able to move if speed was any less than 5. A maximum threshold was not tested, as it was decided that to increase accuracy a slower speed would be utilised.

Motion Experiments

Experiments were conducted for the following set movements: spinning on the spot, drawing a rectangle, drawing a circle, and curve. The log files and video clips for these can be found in the Google Drive link, and all experiment data can be found here[1]. To provide a visual prompt of motion error, a marker pen was taped to the back of Cozmo and placed on a whiteboard. Therefore, these movement experiments were conducted on a different surface to that of the straight-line movements, with the whiteboard material being a lot smoother than the lab tables, allowing for a lot less surface friction, and potentially resulting in a lot more motion drift.

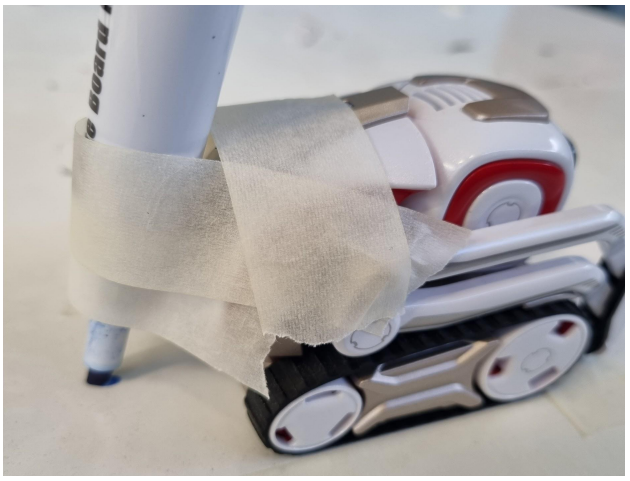

Figure 2

[1] The link to the Google Drive can be found in Appendix A.

Figures 4 and 5 demonstrate Cozmo moving in a circular motion, but at different track speeds, where it moves at a faster speed in Figure 4. The drawing on the board shows the precision of the final image is far greater than the image drawn at a faster speed. Due to the nature of the smooth surface, a faster speed is more likely to result in drifting. Moreover, when wiping the board clean in between usage, it is more likely to be slightly wet, so a dry cloth was used to help reduce any unnecessary surface liquid. These images correlate to Vid-1 and Vid-2 in the repository.

Figure 4

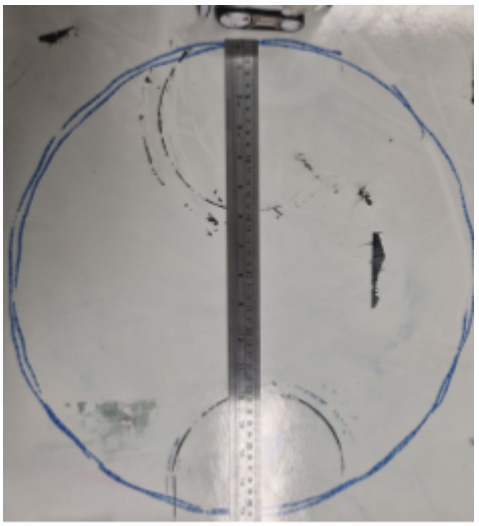

Figure 5

During these experiments, measurements were taken of the distance between the two tracks, giving a value of 35mm. However, the length of the tracks which are in contact with the surface measured 70mm. Moreover, it is worth noting that all measurements of distance were taken from the front of the robot, which might not be the exact position that it measures

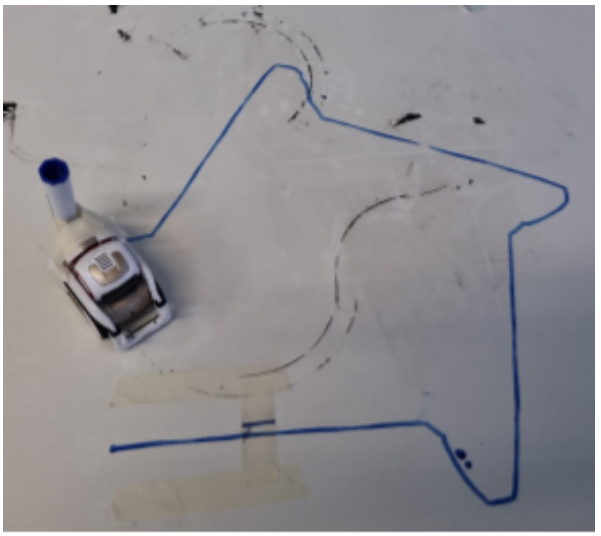

Figure 3

from. Hence, this could be the reason that leads to some discrepancies in its motion variance.

Choice of Motion Models

Considering the specifications of Cozmo, either the velocity or odometry model could be applied. With velocity, we pass in forward and angular values obtained from the inverse kinematics, which represent the instantaneous change of speed in a differential drive robot. Although odometry utilises internal wheel encoders, for our specifications, a velocity model is more appropriate as it provides a probabilistic approach.

Motion Experiments to Target Location

After these initial experiments, the code was developed with the resulting features in mind. Considering the set movement experiments, it was noted that a slower speed would help increase the accuracy. As such, Cozmo was programmed to traverse to a target location on the map; the initial large distance is traveled at a greater speed, and as it nears the target it decreases to a slow speed. These experiments can be observed in Vid-5 and Vid-6. The two aforementioned videos demonstrate Cozmo successfully enacting this experiment, however, Vid-7 demonstrates the exact same code being run on a different Cozmo device. Here it can be observed that the robot overshoots and falls off the table. While watching Cozmo in person, one could see the tracks drifting at certain sections, which then accumulated in it incorrectly rotating, and eventually falling.

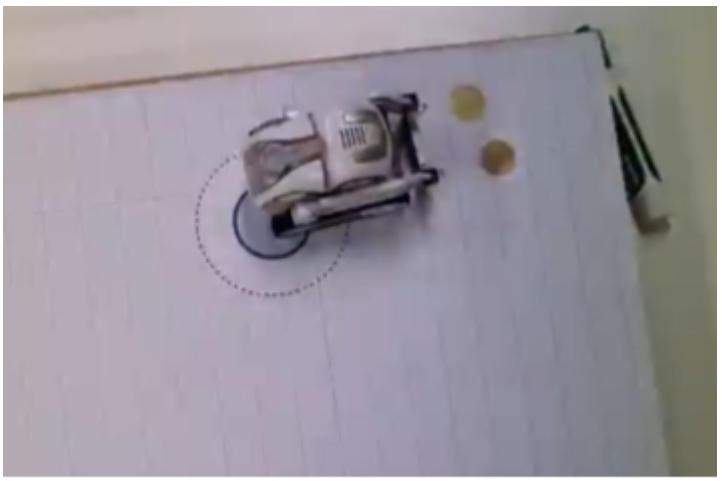

Figure 6

Since a working motion model was implemented, it was utilised in Cozmo's movement towards the target position. From the experiments, and as can be seen in log files, the motion model outputs a high probability of Cozmo's position when moving in a straight line; this means that the motion where both tracks are at the same speed, is fairly precise. However, when it stops and rotates to reposition itself for the next stage of the movement, the probability dramatically decreases. This suggests that the motion when the tracks are different or opposite speeds leads to a decreased likelihood of Cozmo's current pose.

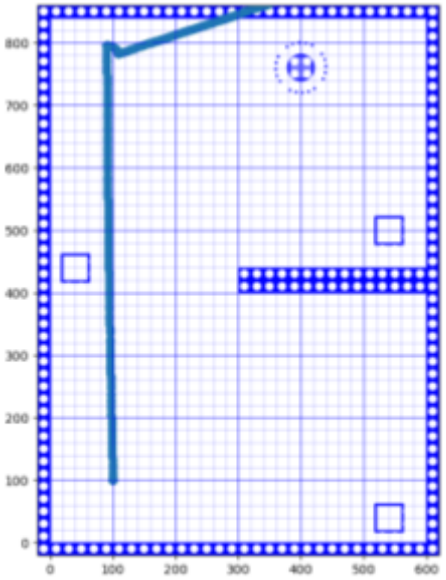

Figure 7

The plot depicted in Figure 7 demonstrates the path taken by Cozmo, where it overshoots on the rotation and drives towards the cliff rather than the target. This plot relates to the video Vid-7.

3. Exploration and Navigation

Path exploration and navigation remain a crucial part of implementing a fully functioning autonomous robot. It enables the robot to traverse through an uncertain environment with a purpose. Furthermore, decision-making when roaming through an uncertain environment allows the robot to gather information and build knowledge about the space it's currently in (Park and Lynch (n.d.)).

Autonomous robots have the potential to be used in environments that would typically be considered dangerous for humans, such as in nuclear reactors, warzones, or sites of natural disasters. For an autonomous system to be able to safely and efficiently maneuver within these cluttered environments is essential, and is the key rationale behind the navigation problem, which leads further into path planning. Moreover, in real-world scenarios these environments will most likely be dynamic, however within the constraints of this assignment, the environment that we will have Cozmo traverse will be static.

There are several available methods for implementing a solution to the navigation problem. For instance, the Visibility Graph generates a topological map with nodes and vertices where the weights can represent the optimal distances between points. One key strategy for solving the navigation problem is to break down the environment into minor sections. One way to do that is through a Voronoi diagram. A Voronoi Diagram creates small conjoining regions in the environment to ensure that the shortest path possible to a specific point is used.

Sensing and Plotting Cubes

The main aim when implementing the decisions made by cozmo is to ensure the existence of navigation competence within Cozmo, and the ability to continue making decisions over a long period of time (Thrun S., 2000). This is certainly achieved when Cozmo senses cubes using its built-in camera sensors. When detected, Cozmo plots the cube at its estimated position which is marked with its cubeID.

Sensing Obstacles and Cliffs

The ability to utilise the gyroscope and the cliff detector is achieved at this particular stage. Cozmo implements a condition in which if the cliff is detected with the help of the cliff sensor, the "drive_wheels_motors" function is initialised which reverses the direction in which the robot is moving at a speed of 100 and then rotates the robot using the built-in gyroscope to a random angle in between 90 and 270. This is implemented to prevent Cozmo from traversing in the same direction every time a cliff is detected. The function in Figure 9 demonstrates this.

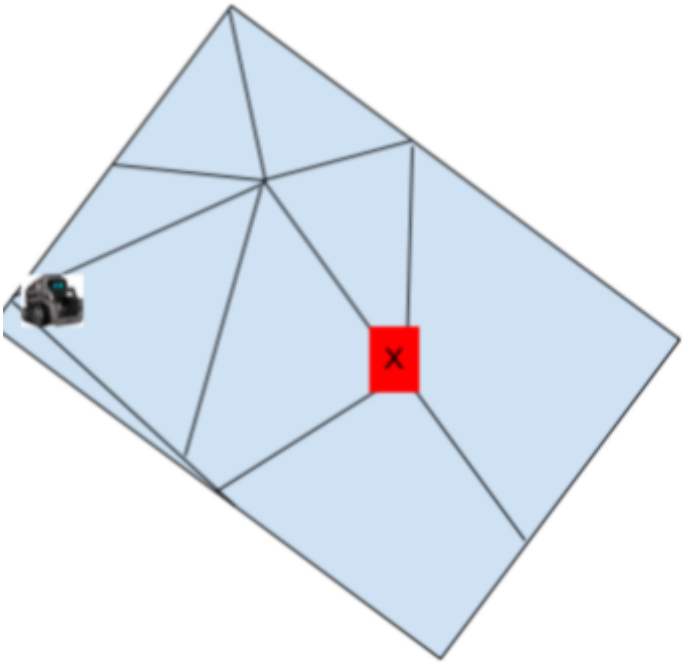

Figure 8

```
if robot.is_cliff_detected:
    robot.stop_all_motors()
    t = time
    print("Cliff detected, rotate 90 degrees to the right and move")
    await asyncio.sleep(1)
    robot.drive_wheel_motors(-100, -100)
    await asyncio.sleep(1)
    robot.drive_wheel_motors(45, -45)
    await asyncio.sleep(2)
```

Figure 9

Looking at the illustration depicted in Figure 8, demonstrates an instance where Cozmo has to get to the target marked on the map, a Voronoi diagram or a generalised Voronoi diagram to be specific, will be essential in its navigation and ensure it uses lines to reach its destination in the shortest route possible.

Random Exploration of Environment

Due to time constraints, the generalised Voronoi diagram was not fully implemented. Instead, an alternative approach was taken in which Cozmo was implemented to navigate and explore the uncertain environment in a randomized fashion and incrementally build its knowledge of the contents in the environment, such as the location of walls and cubes.

```python
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d
import random
from frame2d import Frame2D
import math


def explore(robotpose: Frame2D):
    axis = random.choice([0, 1])
    direction = random.choice([0, 1])
    move = random.randrange(100, 400)

    if axis is 0:
        if direction is 1:
            new_pose = [robotpose.x(), robotpose.y(), (robotpose.angle() + -math.pi/2)] #math.radians(-90))]
            time = 2
        else:
            new_pose = [robotpose.x(), robotpose.y(), (robotpose.angle() + math.pi/2)]
            time = 2

    elif (robotpose.y() + move) < 800:
        new_pose = [robotpose.x(), (robotpose.y() + move), robotpose.angle()]
        time = round((new_pose[1] / 100) * 2)
    else:
        new_pose = [robotpose.x(), robotpose.y(), (robotpose.angle() + math.pi / 2)]
        time = 2


    return new_pose, time
```

Figure 10

Further steps were put in place to ensure the robot moves within a randomised range regarding its orientation to prevent certain instances in which the robot continuously makes a randomised turn, makes a decision and reverts to random exploration. This allowed the robot to move towards a direction when it senses a landmark, make a decision, then revert to it's random exploration.

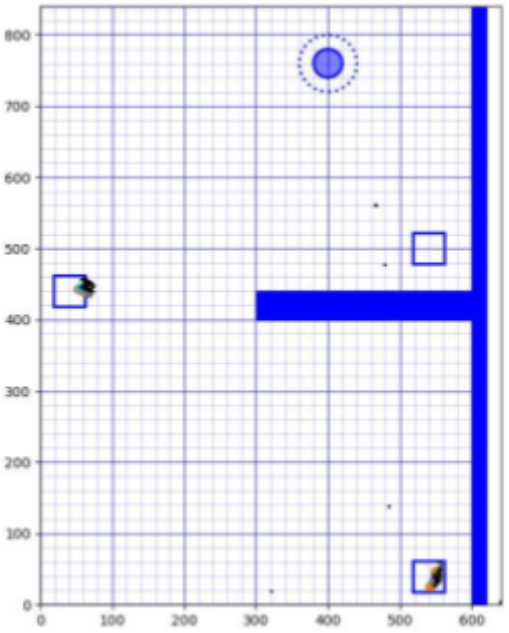

Figure 11

Although the more optimised method was not achieved in time, an alternative such as randomised exploration allows for effective testing of Cozmo when encountering unpredictable situations. Vid-8 provides evidence of the random exploration as described here. Figure 11 is the plot produced by the programme based on the objects detected by Cozmo during its exploration. It can see cubes 1 and 2, and has fairly accurately plotted these; regarding cube 3, it has not plotted anything, but there are small scattered points which could be possible locations that it may have recognised cube 3 to be at.

4. Localisation

Localisation is the process of determining where a machine is at that moment in time and is a vital implementation to any autonomous system with kinematics. This is often achieved using a feature-based method; features can be declared as landmarks. However, these features must be distinct and easily recognisable by the machine's sensors, if a feature is too ambiguous it can lead to misconception and localisation failure.

If a map is provided, landmarks allow the robot to calculate a hypothesis of its general location, also using sensory data provided, an estimated distance between the robot and the landmark is determined. An estimated distance allows the robot to then calculate a distribution of its immediate probable location, if a global map is not available this still gives enough information for the robot to safely traverse the environment. However, it is assumed that an accurate map is available for localisation.

Real-World Example

An example that resembles that of localisation is being placed in an unknown town/city with only street signs and symbols for navigation. In this scenario, the person in question is the mobile robot and his/her eyes are one of the sensors used to recognize the current location inside the unknown city. The person can walk along the streets, observe the different buildings and see the variety of different objects; this allows them to understand their environment, and build an idea of where they are in this location.

If they were then to be plucked from their position and placed into a different location, say a forest, they would be unable to adequately localise themselves on the map. What makes this example interesting is that here the new environment of a forest would have completely unrecognisable features, that being a countless number of similar-looking trees, leading to the person being unable to localise themselves in this environment.

Global Localisation vs Pose Tracking

Localisation can be distinguished in two ways:

Global Localisation - This is a form of local localisation where the robot has no prior knowledge of its current location hence it can be anywhere in the environment.

Pose Tracking -This is another form of localisation where the robot has knowledge of where its starting point is, it then configures further movement given that starting point.

Localisation can be implemented using either of the two techniques explained above, pose tracking for instance can be implemented using a dense representation such as a Gaussian distribution. Global localisation on the other hand cannot be implemented with a gaussian

distribution due to the large level of ambiguity within the environment creating many possibilities of where the robot may actually be. Global localisation can rather implement a sample-based representation which may deteriorate when used with large-scale globalisation which can result in a high amount of computational/memory complexity (Thrun et al., 1999).

Due to the imperfect nature of robotic sensors caused by noise, probabilistic approaches are introduced when implementing localisation. This considers that the sensor readings, as well as the motion model sending the motion commands to the robot, are not 100% perfect.

Implementing probabilistic approaches allows the sensors to display the level of probability through a form of distribution, preferably Gaussian. This is demonstrated below alongside a non-probabilistic model.

Non-Probabilistic Vs Probabilistic Model

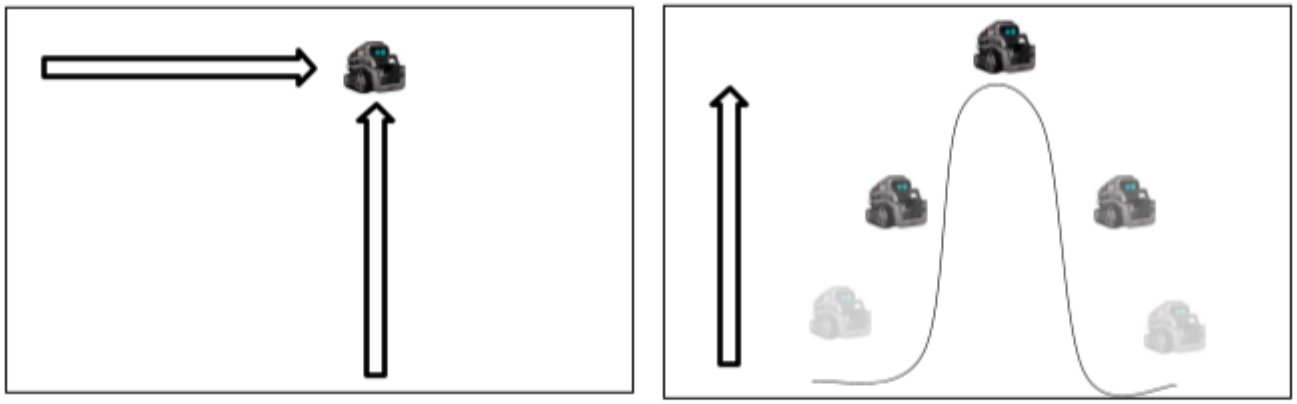

Figure 12

Looking at both diagrams depicted in Figure 12, in a non-probabilistic model Cozmo will be displayed in a fixed position whiles a probabilistic model will display the position as a form of distribution, preferably a Gaussian and the higher the value, the higher the probability of being in that position.

Choice of Methodology

There are several possible approaches to consider when choosing a solution for localisation. Firstly, it is important to consider the possibility of there being either a static or dynamic environment. Static environments are where only the robot moves around, whereas dynamic environments possess objects other than the main robot who have the ability to traverse

around the environment. Furthermore, as Cozmo has known objects - the cubes and walls - these would be considered as “known correspondences”.

One method for localisation is the use of Factor Graphs, which are a form of Bayesian Belief Networks; these offer an adequate level of efficiency compared to other methods, however, they are not as robust as some. Another method is to use an occupancy Grid-based localisation method, however, this can quickly become computationally demanding. As Cozmo has known correspondences, a possible option that would be suitable for this assignment is the use of the Extended Kalman Filter (EKF).

EKF localisation is a form of Markov localisation which is a form of probabilistic state estimation, where it maintains a probability distribution over the entire space for the possible location of the robot. EKF requires that all landmarks are uniquely identifiable, which is the case with Cozmo. Moreover, for instances where the number of landmarks is not too large, EKF can provide an efficient form of localisation.

Implementation

The EKF localisation requires Cozmo’s previous, current positions, state covariance, and velocity. Landmarks seen by Cozmo, a map of the environment, and timestamp are also required data, the current estimated mean/position is then calculated using the equation below.

```
def EKF_localisation(prev_pos, current_pose, covariance, v_vect, seen_landmarks, map, change_time, prev_mew):

    systemVariance = [[20.33, 0, 0], [0, 3.6, 0], [0, 0, 0.001]]

    mew_list = prev_mew

    est_mew = [((-v_vect[0]/v_vect[1]*math.sin(prev_pos.angle())) + (v_vect[0]/v_vect[1]*math.sin(prev_pos.angle()+v_vect[1]*change_time))),
                        ((v_vect[0] / v_vect[1] * math.cos(prev_pos.angle())) - (v_vect[0] / v_vect[1] * math.cos(
                            prev_pos.angle() + v_vect[1] * change_time))), v_vect[1]*change_time]
```

Figure 13

Taylor expansion expands a function into an infinite sum of terms and by doing so improves the approximations. Next congruent transformation is applied to the covariance by using the Taylor expansion and system variances added. Sensor reading was set as 100% accurate with the intention of updating it at a later stage.

```
    taylor_exp = np.array([[1, 0, (((v_vect[0] / v_vect[1]) * math.cos(est_prev_pos.angle())) - (v_vect[0] / v_vect[1]) * math.cos(
        est_prev_pos.angle() + v_vect[1] * change_time))],
        [0, 1, (((v_vect[0]/v_vect[1])*math.sin(est_prev_pos.angle())) - (v_vect[0]/v_vect[1])*math.sin(est_prev_pos.angle()+v_vect[1]*change_time))],
        [0, 0, 1]])


    updated_covariance = np.array(taylor_exp @ covariance @ np.transpose(taylor_exp) + systemVariance) #WHAT IS THIS? (Rt))

    sensorReading = np.array([[1, 0, 0],[0, 1, 0],[0, 0, 1]])
```

Figure 14

For each landmark in the map supplied, the distance between Cozmo’s estimated position and the landmark position is calculated as the variable delta. Conducting matrix

multiplication upon delta and its transpose results in an array of eigenvalues which is then square-rooted to estimate X distance from Cozmo. Next, the variable “z_angle” calculates the Y distance from Cozmos; currently, the landmark orientation is taken from the map itself; it was intended to implement this at a later time. These variables provided an estimated landmark distance from Cozmo’s current location.

Variable H represents a Sensor Jacobian, a Jacobian can be considered a constant and is used to translate between different coordinate systems to represent a measure of change. In this case, it is translating sensor coordinates to map coordinates. Matrix K (Kalman gain) takes the covariance matrix and does the transformation using the jacobian followed by another transformation to later be applied to landmark position estimation.

```
for landmark in map:

    delta = np.array([[map[landmark].pose.x() - est_mew[0]], [map[landmark].pose.y() - est_mew[1]]])

    trans_dist = np.transpose(delta) @ delta # [0, 0]?

    z_angle = math.atan2(delta[0], delta[1]) - est_mew[-1]
    angle = map[landmark].pose.angle()
    est_land_dist.append(np.array([math.sqrt(trans_dist), z_angle, angle]))

    H = np.array(1/trans_dist * [[math.sqrt(trans_dist) * delta[0], -math.sqrt(trans_dist) * delta[1], 0],
                                 [delta[1], delta[0], -1],
                                 [0, 0, 0]])

    matrixK.append((updated_covariance @ np.transpose(H)) @ np.linalg.inv(H.astype(float) @
        updated_covariance.astype(float) @ np.transpose(H).astype(float) + sensorReading.astype(float)))
```

Figure 15

In this section, each seen landmark position is subtracted by the estimated distance from Cozmo and is multiplied against the Kalman Gain matrix. This calculates the variance of each estimation of the distance between Cozmo and the landmark.

```
sum_matk = [0, 0, 0]
for seen in seen_landmarks:
    i = seen.object_id-1
    object = next((j for j in map if j == seen.object_id), None)
    z = np.array([map[object].pose.x(), map[object].pose.y(), map[object].pose.angle()])
    sum_matk = sum_matk + (matrixK[i] @ (z - est_land_dist[i]))

temp2 = [sum_matk[0], sum_matk[1], sum_matk[2]] # due to array issue
mew2 = est_mew + temp2
```

Figure 16

Finally, for each seen landmark, the variance is calculated from the rate of change of the sensor which is multiplied by the covariance to determine the new covariance for the new timestamp. Both the new mean and new covariance are then returned as the next previous versions of both variables during the next cycle.

```
    sum_matk2 = [0, 0, 0]
    for seen in seen_landmarks:
        i = seen.object_id - 1
        sum_matk2 = sum_matk2 + (matrixK[i] * sensor_jacobian[i])

    temp2 = [sum_matk2[0], sum_matk2[1], sum_matk2[2]]
    new_sigma = (np.identity(len(covariance)) - temp2) @ updated_covariance

    mew_list.append(mew2)

    return mew_list, new_sigma
```

Figure 17

Experimentation

Having already experimented with Cozmo conducting set movements, such as a circle or spin, a further round of experiments were conducted that assessed these same movements while utilizing the motion and localisation model. This would allow us to assess and analyse the hypothesis that the probability output from the motion model determines that when track speeds are not the same, the probability decreases.

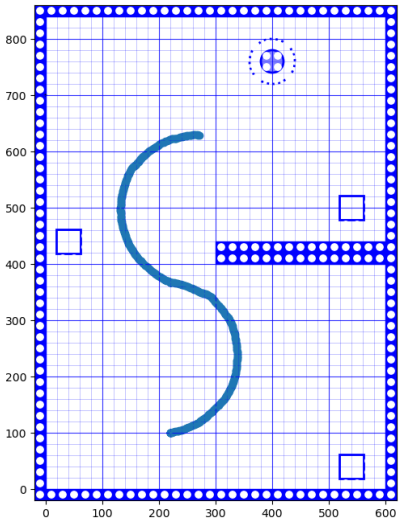

Figure 18

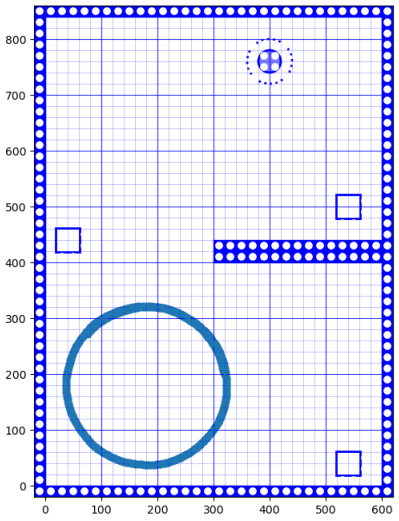

Figure 19

The rationale behind the choice of this experiment is that it helps provide an understanding of how well Cozmo can drive to a predefined path and whether the motion and location model is providing realistic results. However despite previously low motion model probabilities during angular motion these experiments provided reasonably high probabilities, this could be caused by the slow velocity used during these experiments. To pinpoint the source of these strange readings more experimentation and variation are required. These experiments also showed promising localisation results however too few experiments were completed to be certain.

5. SLAM

Simultaneous Localisation and Mapping, as implied by the name, is the act of estimating a map of the environment while simultaneously localising the machine within that map. This makes it a classic ‘chicken and egg’ problem due to the machine's position relying on the map but the map is also dependent on the machine's position (Riisgaard and Blas., (n.d.)). The problem can be simplified by either giving a map then requiring only localisation, or a position which makes mapping a simpler task (Thrun et al., 1999).

SLAM initially starts with little knowledge and a high amount of uncertainty regarding the position or environment of the machine. Bestowing sensory data to a SLAM model provides the opportunity to archive localisation by identifying features from the surrounding environment, utilizing these features a map begins to form. As a map is developed SLAM can make more accurate estimations on the current position of the machine using observations of the landmarks found, and in turn, develop a more accurate model of the environment (Hiebert-Treuer., (n.d.)).

Several mapping methods are often implemented within SLAM; two common examples are "Grid-Based" and "Landmark-Based". Grid-based SLAM produces a topographic map providing more detail of the environment and displaying an estimation of distances between landmarks. This is often shown with mapping models such as Occupancy Grid maps, although withholding more information and providing better visualisation of the environment for users, Grid-based SLAM can quickly become impractical. This kind of mapping is often quite complex, increasing computational demands and therefore increasing the time between the SLAM updates.

If anything is certain it is that real-world applications of these models contain uncertainty; sensor data, motion models, and mapping all contain a level of ambiguity due to noise. Since noise is accumulative, Grid-based SLAM maps can become increasingly inaccurate reducing the value of the extra information provided which can cause maneuverability issues. Because of this, Grid-based SLAM is rarely suited for more than a small simplistic environment (Thrun et al., 1999).

Landmarked-based SLAM requires a topological map that contains only landmark information with an abstract layout. Because this type of map has less information and distances between landmarks are primarily abstract, meaning less complexity, computational requirements are significantly less. This allows less time between SLAM updates, decreasing the risk of higher error rate-making Landmarked-based SLAM an ideal solution for a larger busier environment, and the more favored of the two.

There are several methods to enact SLAM, the main three consist of Extended Kalman Filter SLAM (EKF SLAM), Particle Filter SLAM (PF SLAM), and Graph SLAM. EKF SLAM is a landmark-based method that applies the maximum probability to supplied data using Gaussian noise to imply uncertainty in motion and sensor data.“Historically the earliest, and

perhaps the most influential SLAM algorithm is based on the extended Kalman filter, or EKF" (Thrun et al., 1999).

PF SLAM similar to the EKF uses the landmark-based method, particles are used to hypothesise the state in which the machine is in, each particle represents a possible state. These particles each have a weight of the likelihood that their state is correct and are filtered based on the value. Lower valued particles are discarded leaving the higher probable states to be resampled until the most likely state of the machine is discovered. PF SLAM is a good method for dealing with a non-linear environment and can be extended into a FastSLAM method.

Graph SLAM is another landmark-based method that uses constraints which are the gaussian distributions of the cartesian coordinates. As explained by Giorgio Grissett et al it uses nodes to represent the machine's positions and landmarks. Relative constraints demonstrate the motion from one position to another as well as the distance calculated from sensor data between a landmark and the machine's position (Grisetti et al., (n.d.)).

Choice of model

This project requires successful exploration and mapping of a controlled environment with distinct landmarks, with these specifications it was decided that EKF SLAM was the best fit. EKF SLAM requires regular constant updates within its environment making it an ideal method when dealing with a simple environment, also EKF is effective with a small number of distinct features. EKF SLAM disregards any landmarks it has passed meaning it only processes the landmarks in front of the machine within its detective field, this reduces the computational requirements producing more regular results.

Implementation

The inputs required for EKF SLAM are very similar to EFK localisation, the only difference being these are fairly large matrices that represent the entire environment state, which includes all possible objects and their variables. It is also worth noting that EFK SLAM includes similar methods used in EFK localisation, as it is the culmination of this along with mapping. Hence, the initial algorithm used is fairly similar, with the main difference being that for SLAM these variables are matrices.

F_mask is a large matrix that represents all variables of Cozmo, as well as every possible landmark. Each has 3 variables (x,y, angle), and there are 19 objects, with 1 Cozmo, leading to a matrix of size 60. The system variance was calculated using our experiment data, Ideally, in a fully functional method, this would be updated with each cycle. The variable "est_mew" calculates the new estimated mean for each loop using the previous mean, velocity, and time span since the last update.

```
def SLAM(prev_mew, covariance, v_vect, seen_landmarks, landmarks, change_time):
    F_mask = np.zeros((3, 60))  # 60 as it is 19 x 3

    F_mask[0][0] = 1
    F_mask[1][1] = 1
    F_mask[2][2] = 1

    systemVariance = np.array([[20.33, 0, 0], [0, 3.6, 0], [0, 0, 0.001]])

    temp = np.array([[(-v_vect[0] / v_vect[1] * math.sin(prev_mew[-1][2]) + (
                v_vect[0] / v_vect[1] * math.sin(prev_mew[-1][2] + v_vect[1] * change_time))),
                     ((v_vect[0] / v_vect[1] * math.cos(prev_mew[-1][2])) - (v_vect[0] / v_vect[1] * math.cos(
                         prev_mew[-1][2] + v_vect[1] * change_time))), v_vect[1] * change_time]])

    est_mew = (np.transpose(F_mask) @ temp.T)  # should all be adding the previous mew value (as a vector)
```

Figure 20

The matrix “mat_X” is used within “taylor_exp” for the motion update. It is then manipulating the mean and covariance of the motion model. The identity matrix is also defined in accordance with the dimensions of the covariance, and the transpose of F_mask is also saved in a variable.

```
    I = np.identity(len(covariance))
    FT = np.transpose(F_mask)

    mat_X = np.array(
        [[0, 0, (((v_vect[0] / v_vect[1]) * math.cos(prev_mew[-1][2])) - (v_vect[0] / v_vect[1]) * math.cos(
            prev_mew[-1][2] + v_vect[1] * change_time))],
         [0, 0, (((v_vect[0] / v_vect[1]) * math.sin(prev_mew[-1][2])) - (v_vect[0] / v_vect[1]) * math.sin(
             prev_mew[-1][2] + v_vect[1] * change_time))],
         [0, 0, 0]])

    taylor_exp = I + (FT @ mat_X @ F_mask)

    updated_covariance = taylor_exp @ covariance @ np.transpose(taylor_exp) + np.transpose(F_mask) @ systemVariance @ F_mask

    sensorReading = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

Figure 21

Here, the for loop checks each seen landmark for the indexing of the landmark within its known correspondence list. Then it determines the index for the first row for which the current landmark data is stored; this is used to find the x, y, and angle data for each landmark as it loops. Delta is calculated in the same manner as in EKF localisation, similarly q_k relates to the variable trans_dist, and est_seen relates to est_land_dist. As we are taking sensor readings to be entirely accurate, we locate the correct landmark locations in the F_mask_k matrix and set these to 1.

```
for seen in seen_landmarks:
    j = landmarks[seen.object_id]
    kp = j * 3 + 1

    Pose = Frame2D.fromPose(seen.pose)
    if est_mew[kp][0] == 0:
        est_mew[kp] = Pose.x()
        est_mew[kp + 1] = Pose.y()
        est_mew[kp + 2] = Pose.angle()

    delta = [est_mew[kp] - est_mew[0], est_mew[kp+1] - est_mew[1]]

    q_k = np.transpose(delta) @ delta

    est_seen.append([math.sqrt(q_k), math.atan2(delta[1], delta[0]) - est_mew[2], est_mew[2]])
    F_mask_k[kp][kp] = 1
    F_mask_k[kp + 1][kp + 1] = 1
    F_mask_k[kp + 2][kp + 2] = 1
```

Figure 22

The variable “grape” is a 3 x 6 matrix which is used along with F_mask to create the Sensor Jacobian. Similar to the method in localisation, this is used to determine matrixK, which is the Kalman gain for this landmark.

```
    grape = [[math.sqrt(q_k) * delta[0], -math.sqrt(q_k) * delta[1], 0, -math.sqrt(q_k) * delta[0],
             math.sqrt(q_k) * delta[1], 0],
            [delta[1], delta[0], -1, -delta[1], -delta[0], 0], [0, 0, 0, 0, 0, 1]]

    H = 1/q_k * (grape @ F_mask_k)
    #sensor_jacobean = 1/q_k * [[math.sqrt(q_k) * delta[0], -math.sqrt(q_k) * delta[1], 0, -math.sqrt(q_k) * de
    #               [delta[1], delta[0], -1, -delta[1], -delta[0], 0], [0, 0, 0, 0, 0, 1]] @ F_mask_k

    matrixK.append(updated_covariance @ np.transpose(H) @ np.linalg.inv(H.astype(float)
           @ updated_covariance.astype(float) @ np.transpose(H).astype(float) + sensorReading.astype(float)))
    sensor_jacobian.append(H)
```

Figure 23

Again, these summations for loops follow on from EKF localisation, however, we are using large matrices and updating these.

```
82      sum_matk = [0, 0, 0]
83      for seen in seen_landmarks:
84          Pose = Frame2D.fromPose(seen.pose)
85          z = np.array([Pose.x(), Pose.y(), Pose.angle()])
86          sum_matk = sum_matk + matrixK[i] * (z - est_seen[i])
87          i += 1
88
89      mew2 = est_mew + sum_matk
90
91      i = 0
92      sum_matk2 = np.array([[0, 0, 0], [0, 0, 0], [0, 0, 0]])
93      for seen in seen_landmarks:
94          sum_matk2 = sum_matk2 + matrixK[i] @ sensor_jacobian[i]
95          i += 1
96
97      new_sigma = (I - sum_matk2) @ updated_covariance
98
99      return mew2, new_sigma
```

Figure 24

Experimentation

Due to time constraints, few experiments were completed with a working EKF SLAM model, Figure 25 is a plotted result of Cozmo's path within the designed map. This test applied SLAM to the implemented hard target experiment, as shown the motion was very different from the original experiment. The cause of this appeared to be the high computational requirements of the EKF SLAM, to compensate for this the time given to complete each stage of motion was decreased. Despite SLAM producing seemingly accurate results (Figure 26) of Cozmo's finishing position and the cube position along its path, as expected the system is rather delayed.

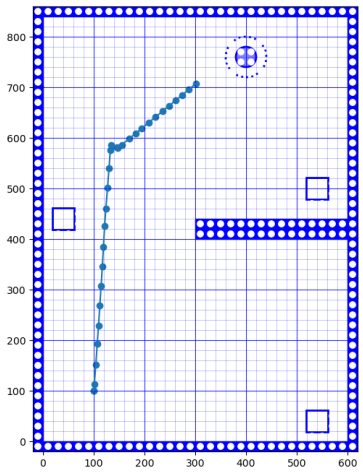

Figure 25

```
(13, [[[631.79120924]) [625.98338791]) [647.85300577])]
 [[260.53645635]) [-193.83635749]) [45.69987982])]
 [[-0.035218781) [0.22567754]) [0.14084771])]
 [[0.]) [0.]) [0.])]
 [[0.]) [0.]) [0.])]
 [[0.]) [0.]) [0.])]
 [[0.]) [0.]) [0.])]
 [[0.]) [0.]) [0.])]
 [[0.]) [0.]) [0.])]
 [[0.]) [0.]) [0.])]
 [[342.92636108]) [342.92636108]) [342.92636108])]
 [[17.36328697]) [17.36328697]) [17.36328697])]
 [[-0.24064523]) [-0.24064523]) [-0.24064523])]
```

Figure 26

6. Conclusion

From conducting initial experiments, it was observed that the motion of Cozmo is fairly consistent. However, upon further experimentation using a different Cozmo device, it was noted that it was far less accurate considering it was running the same code. Therefore, if we were in a position to do this again, it would be vital to ensure that the Cozmo unit being used is kept consistent. Moreover, the use of different units could be leading to different internal errors and variances, as well as system battery life having an impact on the ability of the robot to be able to move at certain speeds.

The design and implementation of a motion model allowed us to observe the likelihood of the motion accuracy, it was noted that motion along straight paths would yield a higher probability. Our prior hypothesis stated that rotational probability was low, however, upon further experimentation, the probability values appeared to increase with a slower angular motion. To further improve the motion model, a variety of different motions would be applied to show refinements that could be made.

The implementation of the path exploration and navigation allows the team to utilise the kinematics implemented prior and use it to traverse the environment while plotting landmarks. Although annotating how the robot traverses randomly in an uncertain environment is one way to judge the effectiveness of Cozmo, given more time in the future a much-optimised method such as the generalised Voronoi diagram can be implemented. Be as it may, the unpredictable fashion of the previous exploration method is no longer viable in this new diagram; implementing a Voronoi diagram will allow the measurement of real-time decision making and path planning, that will result in Cozmo sensing and interacting with landmarks in the environment in the shortest time possible.

As explained in localisation and SLAM implementation certain variables were pre-defined, such as the sensor reading and system variance. Further work upon this project would see that these variables are initialised accurately using experimental data and recalculated each cycle with all previous data. Also, both models currently estimate only cartesian coordinates, orientation is kept as the initial reading returned by the sensor; this was intentional. Primarily we wanted to have both localisation and SLAM working efficiently with cartesian coordinates, after which orientation would have been included. Despite these constraints, the experiments showed that both EKF localisation and EKF SLAM implementation successfully run and produce reasonable results. However, continuing from here more experiments would need to be enacted to ensure these findings, including some debugging to reduce computational time. One method of achieving this could be to introduce threading for these models, and applying a refined version of the EKF SLAM model to an effective refined exploration and navigation method would be the final goal.

Overall, this project has developed our understanding and knowledge of robotics, autonomous systems, and probabilistic models. The project was challenging but allowed room for each of us to develop our understanding of the program subject matter. If we were to reapproach this project, conducting experiments in tandem with implementation would increase understanding of the efficiency of our models, therefore providing us with greater knowledge to refine them further.

7. References

Thrun S, Burgard W and Fox D (1999) *PROBABILISTIC ROBOTICS*.

Anon ((n.d.)) Prediction of Position Fig 5.2 General schematic for mobile robot Localisation.

Anon ((n.d.)) Probabilistic Motion Models Introduction to Mobile Robotics. .

Durrant-Whyte H and Bailey T ((n.d.)) Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms.

Grisetti G, Kümmerle R, Stachniss C and Burgard W ((n.d.)) A Tutorial on Graph-Based SLAM. Available at: http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti10titsmag.pdf

Hiebert-Treuer B ((n.d.)) An Introduction to Robot SLAM (Simultaneous Localisation And Mapping).

Olson CF (2000) Probabilistic Self-Localisation for Mobile Robots. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION* 16(1): 55.

Park FC and Lynch KM ((n.d.)) INTRODUCTION TO ROBOTICS MECHANICS, PLANNING, AND CONTROL.

Riisgaard S and Blas MR ((n.d.)) SLAM for Dummies A Tutorial Approach to Simultaneous Localization and Mapping By the “dummies.”

Thrun S (2000) Probabilistic Algorithms in Robotics. *AI Magazine* 21(4): 93–93. Available at: https://ojs.aaai.org/index.php/aimagazine/article/view/1534 (accessed 09/12/21).

Waldron K and Schmiedeler J (2007) Handbook of Robotics Chapter 1: Kinematics.

Appendix A

The Google Drive link below provides the videos as mentioned throughout this report. A second folder in this drive contains all log files, images and any other supplementary information for the experiments.

Google Drive Link: https://drive.google.com/drive/folders/0APyh4cRm9t7-Uk9PVA
People with access to this link: Alex Rast, Mattias Rolf, Daniel Addo, Methuselah Singh, Sam Trowbridge.