**CMU506 – Robotics**

**Lab Report:**

**Lab 2: Accurate Robot Motion**

|  |  |
| --- | --- |
| Team Members: | Hannah, Ethan and Teodora. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Self-Mark** | | **Rationale** |
| **Self`** | **Total** |
| Understand and Explain the Robot Program |  | **4** |  |
| Distance and Rotation Calibration for Accurate Driving in a 1m Square |  | **10** |  |
| Experiment to Measure Robot Accuracy |  | **8** |  |
| Calculating a Covariance Matrix |  | **5** |  |
| Experiment with Modifying Your Robot to Make it More Accurate |  | **8** |  |
| **Total** |  | **35** |  |

Understand and Explain the Robot Program (4 marks)

|  |
| --- |
| * Gaussian – generates a random sample from a gaussian/normal distribution * resetBase – returns the robot to a starting location, by locating all the objects in the simulation, and then resetting all the objects. * createRandomBumpyFloor – resets, then creates a random bumpy floor for the simulation, to improve the testing conditions – no surface in real life is 100% ‘smooth’ * sysCall\_init – ran singular time at start - code controlling movement of robot – lists movement steps for the robot to take, records starting positions, calls on ‘createRandomBumpyFloor’ to generate the next bumpy floor being used. * isCurrentTargetAchieved – checks to see if the robot has stopped based on linear velocity, and angular velocity * sysCall\_actuation – checks where the robot has stopped, and if it has completed the prior step correctly, using motor angle targets, step counters and more. * sysCall\_cleanup – resets and closes the simulation. |

Distance and Rotation Calibration for Accurate Driving in a 1m Square (10 marks)

|  |
| --- |
| Here we have the steps listed out that control the robot’s movement. In the end, we decided on setting the turn angle to ‘math.rad(28)’, the forward distance to 0.6, ‘motorAnglePerMeter’ to 41.5, and ‘motorAngleMeter’ to 10. We came up with these values, based entirely on experimentation.    We started by increasing the ‘stepList’ from 4 commands to 17. We did this so we could specifically detail what steps we wanted the robot to take, rather than relying on a loop. We had to do this because when we started the initial setup, we found rather than the robot sitting idle, it would fall through the floor. This behavior was unexpected, and we weren’t too sure what was causing the issue. We ended up very slightly altering the ‘bumpyfloor’ generation, and increasing the size of the ‘stepList’, and this managed to solve the problem.  The next step was setting the values to an arbitrary number and seeing what happened. The aim was to have the robot travel in a square, by moving forward a meter, and then turning left. This would the repeat until the robot was back in the starting position. This would be repeated 10 times. We would then change the angles, based on where the robot would end up, so if it was ending up to the left of the starting point, then we would increase the rad and angle per rad, and if it ended up on the left, we would lower the values. We kept doing this, repeating trial and error, until we were at a point where it would consistently end up very near the starting point |

Experiment to Measure Robot Accuracy (8marks)

|  |
| --- |
| We implemented code to trace the paths that the robot would take, and then display the results of all the paths in a graph. The graph produced above shows that whilst the robot has the potential to move in a perfect square, the reality is that the robot would usually end up somewhere close the starting point, but not solely on one side which an incorrect calibration would end up. It would be very difficult to be able to get it to move in a perfect square. There are a few sources of inaccuracies that could be causing this. This can be down to either slight changes each time a new ‘BumpyFloor’ is generated, or it could be down to some slight error with the values that we have set, however, this would be very difficult and time consuming to fully fine-tune |

Calculating a Covariance Matrix (5 marks)

|  |
| --- |
| Above, we have the code for creating a covariance matrix. The covariance matrix is an explicit measure of the scatter range, or uncertainty in the motion of the robot. The covariance matrix is a matrix of 4 values, symmetrical diagonally, made up of the coordinates of the mean final location. The code above is split into 4 sections, the first two sections are responsible for calculating the variables required to get the 4 values, the third section is responsible for calculating and assigning names to the mean final location coordinates, and the final section prints the covariance matrix once the simulation has finished running. A screenshot of the final output can be seen below.    Where we can see the covariance matrix printed out at the bottom |

Experiment with Modifying Your Robot to Make it More Accurate (8 marks)

|  |
| --- |
| The first thing that was done when making it more accurate was to remove the bumpy floor. We did this so that we could see what our code is doing without external interference. To do this, we set the ‘heightfieldnoise’ to 0, this would still generate a floor, but it would be flat. You can see the results below    Here, you can see the graph plotted is much more uniform and neater than it was before. You can also see the covariance matrix is much smaller than it was previously. It is still however, not a ‘perfect’ square. There is some very slight differences, but they all seem to end up on the right of the starting point, meaning that the angle it is turning is too sharp. Taking the turning rad. Down to 27 means that it isn’t turning enough, and always ends on the left of the starting point. In the end, without the bumpy floor, we managed to get it to this point    Which ends up being less than a centimetre difference between the starting point, and where the robot stopped. Turning the bumpy floor back on did case a good bit of error, but not as much as previously |