Lab 1 – Kinematics

Austin Holmes – U#31858364

Control of Mobile Robots – Section 001F23

INTRODUCTION

The first lab of the control of mobile robots series explores the kinematic method of robot control, developing and demonstrating the ability to direct the motion of a mobile robot by predicting necessary motor outputs and actively monitoring distance traveled to ascertain goal achievement. This report documents the difficulties and achievements associated with my attempt to do this using FAIRIS-Lite within the Webots simulation environment.

TASK I – WAYPOINT NAVIGATION

The first and only primary task presented in this lab is navigation of a series of waypoints which represent a valid path through the simulated environment. This navigation task is achieved by way of three (3) primary motion functions: linear motion, arc traversal, and static rotation. The robot must also output its estimated position, a function linked to time.

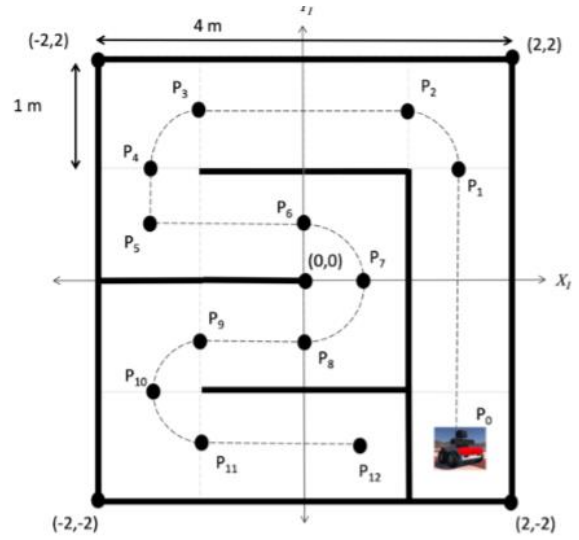

Figure 1 – Visualization of Path Given by Waypoints

KEY VALUES

The implemented controller relies on many parameters which potently influence behavioral decisions such as the speed or direction of travel. This list is not exhaustive and covers only values which will be referenced further.

A. Speed Preference

Speed preference, set separately for linear, arching, and rotational motions, determines the rotational speed of the motors, and thus the movement speed of the robot.

B. Precision Preference

Precision preference, considered for both position and bearing information, represents the allowed margin for error in ending movements.

KEY FUNCTIONS

There are also a number of critical utility functions which drive the execution of decision and movement functions. This list is not exhaustive and mentions only functions which will be referenced further.

A. Move Until

The move_until utility function underpins all functions which govern the cartesian motion of the robot, accepting both a starting and intended distance of travel as minimum inputs. This function continues to execute at every timestep until the distance traveled meets the intended distance to the applicable level of precision. It may also accept differential wheel velocities in order to properly calculate distance over an arching path. This function relies heavily on the following distance calculation, where d is distance, r is wheel

radius, and c_{avg} is the average encoder reading (relative to starting position):

$$d = rc_{avg}$$

B. Update Estimates

The update_estimates function is critical both to diagnostics as well as behavior. Executed every timestep, this self-contained function utilizes the compass provided by the Intertial Measurement Unit (IMU) in addition to the wheel encoders in order to determine the distance traveled since the previous update and trigonometrically apply that distance and angle to update the internal position estimate of the robot. Those functions, where d is distance since last estimate, θ is the heading of the robot, and x and y represent their respective estimated coordinates, are as follows:

$$x = x + dcos(\theta)$$

$$y = x + dsin(\theta)$$

MOTION FUNCTIONS

These primary motion functions control all movement performed by the robot. Inputs are processed such that all movements are possible (e.g. no negative distance) and minimal (e.g. bounding angles within 180 degrees of zero (0)). It is also of note that a braking functionality has been implemented which lowers the velocity of the robot as it approaches the next waypoint in order to mitigate errors induced by jarring changes in velocity.

A. Linear Motion

Linear motion, the simplest of all requisite motion types, is accomplished simply by setting both motor speeds to the preference parameter and utilizing the move_until function to control stoppage. When estimating the time taken to accomplish such a movement, a simple formula where d is distance, t is time, and s is speed is used:

$$t = d/s$$

B. Arching Motion

Movement in an arch, while similar in conceptual basis to linear motion, requires the precise calculation of inner and outer wheel speed ratio. This function depends on inputs of arclength and radius, and calculates wheel velocities, where v_o is the outer wheel velocity, v_i is the inner wheel velocity, r is the radius, and a is the axel length as follows:

$$v_o = r + \frac{a}{2}$$

$$v_i = r - \frac{a}{2}$$

Results are then normalized according to arching speed preference s:

$$v_o = \frac{sv_o}{\frac{v_o + v_i}{2}}$$

$$v_i = \frac{sv_i}{\frac{v_o + v_i}{2}}$$

These velocities are then maintained as in linear motion by move_until such that the input arclength is traversed.

C. Static Rotation

In-place rotation, a simple matter of adjusting the bearing of the robot, is accomplished by setting each of the motors of the robot to the rotational speed preference with opposing sides opposing in direction. If the rotation is clockwise, the left motors will be forward, and the opposite will be true for counterclockwise rotation. This motion is continued until the compass reads a bearing within the rotational precision preference. No predictions are made.

NAVIGATION SCRIPT

As the program must accept a simple array of waypoints as input, the control script must contain some means of determining the intended motion function. In this case that is done by a simple navigation script which is sensitive to x and y distances between current and future waypoints. In cases where the

distance to be traveled along one axis appears trivial according to linear precision preference, linear movement will be employed along the other more significant axis (after rotating to the proper heading). If both axes require significant movement, an arch will be made in the minimal direction. The minimal direction for arches as well as rotations in preparation for linear motion are calculated trigonometrically, where Δx is x-axis distance, Δy is y-axis distance, and θ is the angle of travel:

$$\theta = atan2(\Delta y, \Delta x)$$

Note: atan2(a,b) represents the arctangent function arctan(a/b) with inbuilt corrections for negative-value-induced quadrant offsets.

DIAGNOSTIC OUTPUT

During navigation to every waypoint, the robot outputs a variety of diagnostic data.

A. Next Waypoint Difference

During and before navigation, the robot prints the distance to be traveled along both axes as well as the angle to be traveled along. These values are all calculated by the aforementioned navigation script.

B. Velocity and Estimated Time of Arrival (ETA)

The speed and ETA values, are provided by the motion functions themselves – each supplies its wheel velocities and intended distance of travel to a print function which estimates the time of travel based on that data according to the formula below, where t is travel time, d is distance to be travelled, and v is the average wheel velocity:

$$t = d/v$$

Additional time is then added in the same fashion to account for any braking behavior before outputting both the input and resulting data.

C. Live Pose

The live heading and position estimate data is provided by the aforementioned update_estimates function and is refreshed every timestep in-place for ease of viewing along with the other data.

CHALLENGES

The most impactful challenge of this project is most certainly accounting for imprecision. Especially when executing large, arching motions, small imperfections in actual distance travelled grow into a significant cause for error which must be accounted for. After first getting around this by hard-coding small offsets, I shifted the responsibility of handling this to the navigation script such that subsequent motions first correct for angle, enabling the robot to dynamically determine the ideal path in linear cases.

Another major issue arises from the navigation script concerning how to decide whether to make a linear or arching path to the next waypoint. My temporary solution is to only arch in cases where significant motion is required in both axes, but that is very sensitive to the waypoints provided.

CONCLUSION

Accounting for environmental error is easily the most critical consideration in developing a robust mobile robot control system. In that, enabling the robot to make as many independent decisions as possible appears to be a potent strategy.

NOTES

Within the program, wheel velocities are assumed to be angular when stored or retrieved, meaning conversions must be made in cases where the linear velocity of the robot is needed, as specified.