HW 1 Report

Kinematic Model

We used the general kinematic model introduced in class. Specifically, we used the basic kinematic model where α was ± π/4 or 3π/4 for each wheel, β was ± π/2, and γ was ± π/4, pictured below, and derived the inverse to find angular velocities given robot velocities v_x, $v_{y,}$, and ω_z:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} & -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}$$

i	Wheels	α_i	β_i	γ_i	l_i	l_{ix}	l_{iy}
0	1sw	$\pi/4$	$\pi/2$	$-\pi/4$	l	l_x	l_y
1	2sw	$-\pi/4$	$-\pi/2$	$\pi/4$	l	l_x	l_y
2	3sw	$3\pi/4$	$\pi/2$	$\pi/4$	l	l_x	l_y
3	4sw	$-3\pi/4$	$-\pi/2$	$-\pi/4$	l	l_x	l_y

Our navigation algorithm consisted of rotation in place and translation. We extracted the world frame coordinates (x, y, θ) given in waypoints.txt, then split up joint transformations into translations or rotations. Then, using an arbitrary time of 4 seconds, we calculated (Δx, Δy, Δθ) for each transformation with respect to the robot's body frame, after converting each of the waypoints (x, y, θ) from the world frame to the robot's body frame. We then calculated the necessary velocities for the robot v_x, v_y, and ω_z for each transformation. The table below shows our transformations and the associated v_x, v_y, and ω_z velocities. After calculating these velocities, we plugged them into the model to retrieve the angular velocity for each of the wheels ω_i as a matrix. As our final step, we converted the angular velocities to the integer motor velocities using the equations derived during our calibration (found in the calibration section below) and iterated through each waypoint A → B.

Applied Transformations:

A (x, y, θ)	**B (x, y, θ)**	**(Δx, Δy, Δθ)**	**Time (s)**	**v_x (m/s)**	**v_y (m/s)**	**ω_z (rad/s)**
0,0,0	-1,0,0	(-1,0,0)	4	-0.2500	0	0
-1,0,0	-1,0,1.57	(0,0,1.57)	4	0	0	0.2617
-1,0,1.57	-1,1,1.57	(0,1,0)	4	0.2500	0	0
-1,1,1.57	-1,1,0	(0,0,-1.57)	4	0	0	-0.2617

-1,1,0	-2,1,0	(-1,0,0)	4	-0.2500	0	0
-2,1,0	-2,1,-1.57	(0,0,-1.57)	4	0	0	-0.2617
-2,1,-1.57	-2,2,-1.57	(0,1,0)	4	-0.2500	0	0
-2,2,-1.57	-2,2,-0.78	(0,0,0.79)	4	0	0	0.1317
-2,2,-0.78	-1,1,-0.78	(√2,0,0)	4	0.3535	0	0
-1,1,-0.78	-1,1,0	(0,0,-0.78)	4	0	0	0.1317
-1,1,0	0,0,0	(1,1,0)	4	0.2500	-0.25	0

Calibration

Our calibration involved a few steps. First, we measured l_x (half the distance between front wheels), l_y (half the distance between front and back wheels), and r (the radius of the wheel), which are necessary for the kinematic model. $l_x = 0.069$ m, $l_y = 0.057$ m, r = 0.029 m. Then we corrected the sign functions in mpi_control.py after testing it so the functions carSlide, carRotate, and carStraight performed as expected (we only used carStraight in the calibration part). We also fixed the ports for the wheels, which were flipped around.

Next, we measured the minimum motor velocity (integer) for each wheel by running setFourMotors with one wheel velocity ω_i at a time. Although it was sometimes inconsistent, due to the wheel occasionally getting jammed, the minimum motor velocity was around 20 for each wheel.

The last step of our calibration was to calculate the rotations/s for each wheel. To do so, we selected 3 motor velocity points that were greater than the minimum motor velocity. They were on the slower side for the sake of accuracy with counting. Then we ran carStraight for about 10 seconds while we measured how many times the wheel rotated in those 10 seconds. From these points, we derived equations for each wheel to convert angular velocity in radians/s into the motor velocity command, an integer, for each wheel.

Wheel	**Motor Velocity**	**Rotations/s**	**Radians/s**
Front left	25	0.55	3.46
	30	0.60	3.77
	35	0.80	5.03
Front right	25	0.53	3.33
	30	0.65	4.08
	35	0.80	5.03

Back left	25	0.55	3.46
	30	0.68	4.27
	35	0.80	5.03
Back right	25	0.55	3.46
	30	0.65	4.08
	35	0.80	5.03

- Front left wheel equation: $v = 5.68w + 6.79$
- Front right wheel equation: $v = 5.87w + 5.66$
- Back left wheel equation: $v = 6.36w + 2.95$
- Back right wheel equation: $v = 6.28w + 3.68$

Algorithm Performance

Our algorithm performed adequately. Reaching the first waypoint through translation, the robot performed extremely well. The subsequent rotation was around the provided $\pi/2$ radians. We noted that the rotation was more than $\pi/2$ radians (closer to π) when we set it to θ_b (theta expected) - θ_a (theta current) / Δt, but multiplying the rate by a factor of 1/1.5 allowed us to have a more accurate result for the rotations. After the first rotation, the robot translated in the correct direction, but went far past the second waypoint. The distance of the next translation was fine, but the position was definitely above the second waypoint since the previous translation went too far. After that, the first diagonal translation was too far, while the second one was not far enough. It could also be due to a difference in friction from multiple factors. One factor could be that the surface had vertical planks, so the robot moved faster going parallel to the planks. Another could be the difference in friction between translating diagonally with different orientations (e.g. an orientation of 0 radians needs more speed to go diagonally than an orientation parallel to the diagonal). This makes our calibration not perfectly applicable to every case. Overall, our algorithm had decent performance, with a few errors that can possibly be fixed using a PID controller.

YouTube Video Link: https://www.youtube.com/watch?v=Ffy2ovBMyBo

*Note: To run our code, you can simply run “python3 model.py” in the rb5_ros2_control directory.