Laboratory Experiment II

-Robotics Course-

6. Control Methods for Differential Drive Type Robot

田所・昆陽・ 多田隈研究室

Tadokoro, Konyo, Tadakuma Lab

Experiment Date: 13 January 2023

Name: Farros Alferro

Student ID: C0TB1706

Group F

Control Methods for Differential Drive Type Robot

Farros Alferro [1] **Shotaro Kojima** [2]

1. Introduction

Because of its simplicity, differential drive is used by many mobile robots. It comprises two driving wheels positioned on a shared axis, and each wheel can be driven forward or backward separately. It can thus alter its direction by varying the relative rate of rotation of its wheels, eliminating the need for an additional steering motion [1].

In this experiment, we will study the principles underlying the autonomous navigation of a differential drive type robot and then test our comprehension by developing software and navigating the robot. Furthermore, we will apply what we have learned to create an autonomous robot capable of reaching the specified goal point on its own [2].

2. Principles

2.1. Differential Drive Type

The mobile robot in this experiment includes two drive wheels, each with its own motor that steers by altering the rotational speed of the motors (see Figure 1). This robot's configuration is known as a differential drive. The differential drive is distinct from car-like drive, which has one motor for two wheels and another for steering. Because of the following benefits over the car-like drive type, differential drives are often employed for mobile robots: (a) they may rotate without altering position, and (b) the design is concise because no steering system is required. Nevertheless, it has drawbacks, such as a lack of steadiness when moving straight.

2.2. Kinematics

The velocity of the mobile robot would be the control reference for controlling its mobility. On the other hand, the rotation of the wheels is necessary to move the robot. As a result, the mobile robot's control must include a model that

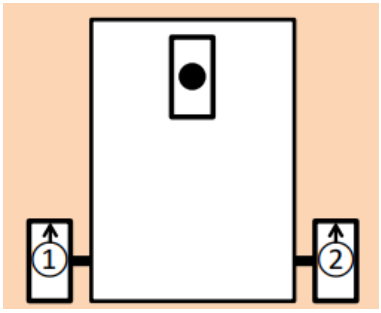


Figure 1. Differential wheeled robot [3].

depicts the relationship between the robot's velocity and the rotational velocity of the wheels. It is called "forward kinematics" or simply "kinematics" to describe the former using the latter, while it is called "inverse kinematics" to describe the latter using the former. The (forward) kinematics of the differential drive type robot can be described using the equations below:

$$v = \frac{v_r + v_l}{2} = \frac{r(\omega_r + \omega_l)}{2} \tag{1}$$

$$\omega = \frac{v_r - v_l}{T} = \frac{r(\omega_r - \omega_l)}{2} \tag{2}$$

where v and ω denote the linear and angular velocity of the middle point of the wheels, respectively. Note that the subscript $*_r$ indicates the right wheel while $*_l$ indicates the left wheel. Lastly, the radius of the wheels is denoted as r, and the separation of the wheels as T. The diagram of these variables is shown in Figure 2.

By integrating its velocity, we can also determine the robot's pose (position and orientation). Odometry is the name given to this integration. Odometry is commonly employed as a basic position estimation method for mobile robots, despite its limitations due to the lack of consideration for slip between the wheels and the ground. The odometry is described with the following equations:

[1]Computer Vision Lab., Tohoku University, Sendai, Japan [2] Human-Robot Informatics Lab., Tohoku University, Sendai, Japan. Correspondence to: Farros Alferro <farros.alferro.t3@dc.tohoku.ac.jp>, Shotaro Kojima <kojima@rm.is.tohoku.ac.jp>.

Theme 6 Report on *Laboratory Experiment II* which was done on January 13th, 2023.

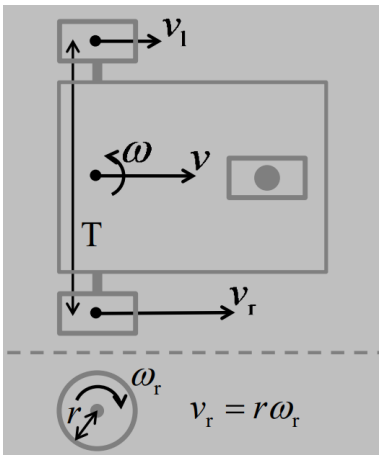


Figure 2. Robot's kinematics [3].

$$x = \int_{t_o}^{t} v \cos\theta \, dt + x_0 \tag{3}$$

$$y = \int_{t_o}^{t} v \sin\theta \, dt + y_0 \tag{4}$$

$$\theta = \int_{t_o}^{t} \omega \, dt + \theta_0 \tag{5}$$

where (x, y) denotes the right-handed coordinate system, θ is the angle between the x-axis and the robot, and subscript $*_0$ indicates the initial value at $t = t_0$ (see Figure 3).

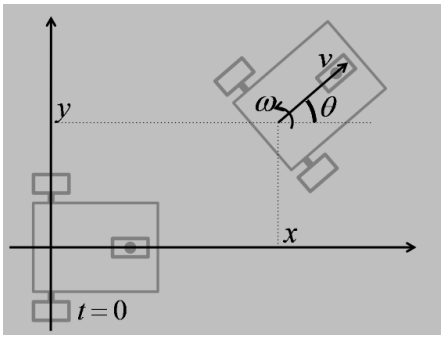


Figure 3. Robot's position and orientation [3].

2.3. Path Following Control

Suppose we use kinematics and odometry to estimate the robot's posture and inverse kinematics to accept the robot's velocity as a reference value. In that instance, we can direct the robot along a predefined course. "Path following control" is a common navigation control mechanism [4]. To reduce the angular and Euclidean divergence between the path and the robot, the path following control sets the reference value of the robot's angular velocity using the following equations:

$$v^{ref} = max(V - c|\omega|, 0) \tag{6}$$

$$\left(\frac{d\omega}{dt}\right)^{ref} = -k_\omega \omega - k_\phi \phi - k_d d \tag{7}$$

where the superscript $*^{ref}$ is the desired value, and V is the maximum linear velocity of the robot. Moreover, ϕ and d are the angular and Euclidean differences between the path and the robot (see Figure 4).

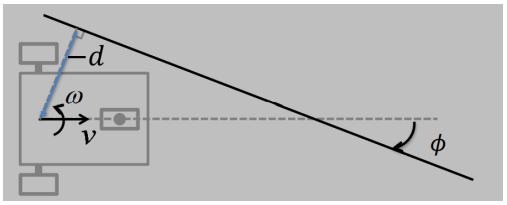


Figure 4. Path following control [3].

2.4. Costmap and Path Planning

If we notify the goal point and provide the robot with environmental information, the robot can plot its journey to the provided goal point. Furthermore, if the robot can follow the prescribed path, it can reach the objective location automatically.

"Costmap" is a common approach for representing environmental information needed in path planning. The costmap displays space as grids and retains the numerical score of each grid (cost). The cost reflects the difficulty of positioning the robot's center on the grid (greater cost means greater difficulty). For example, we can apply the following guidelines to grid costs:

- Infinite cost to grids that contains an obstacle
- Infinite cost to grids that close to the obstacle and must cause a collision with it
- Positive cost to other grids according to navigation difficulty or priority

Once the costmap is provided, the robot may plan a suitable course to the objective location by minimizing the sum of costs along the way (see Figure 5). We can use well-known path-finding algorithms such as Djikstra's algorithm, which guarantees to find a path with the lowest costs, or the A* algorithm, which finds a path with low costs in a shorter time.

2.5. Example Software for Autonomous Navigation

We can design software for the differential drive type robot, which can autonomously plan and follow a path to a goal

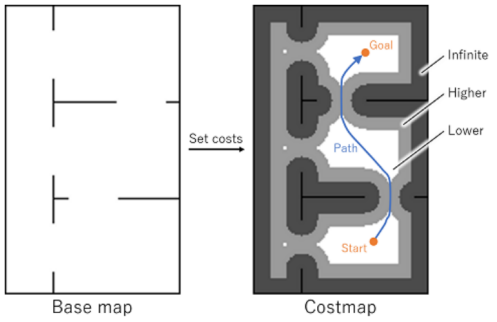


Figure 5. Costmap and path planning [2].

point supplied by the human user, by modularizing each function discussed in Sections 2.2 to 2.4 and merging these modules. Combining simple software components, like this program, is a popular strategy in robotic development for realizing an advanced function (Figure 6).

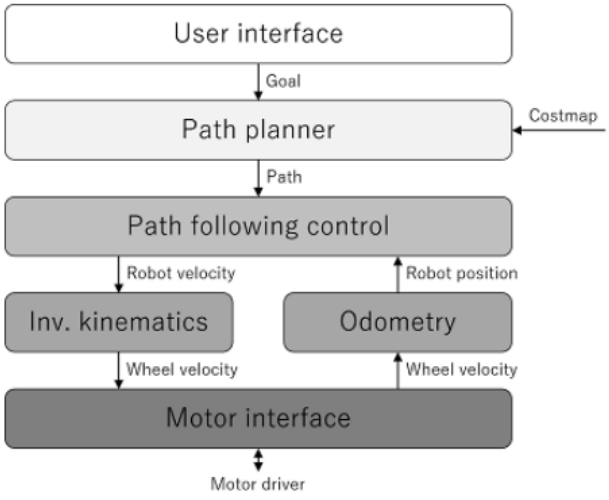


Figure 6. Autonomous navigation software for this experiment [2].

3. Experiment

3.1. Setup

The instruments for the experiment are a portable robot [5] and a laptop computer running user interface software. The robot and the laptop are linked through wireless LAN.

The robot's left and right wheels are driven independently by DC motors. Each DC motor has rotary encoders to measure the rotational speed of a wheel, which is essential in the kinematics calculation. Furthermore, because a feedback control for the rotational speed of the wheels is provided in the motor driver's firmware, the robot may realize the reference rotational velocity of the wheels. We employ two servo-motors in this experiment, each with a motor, encoder, and motor driver.

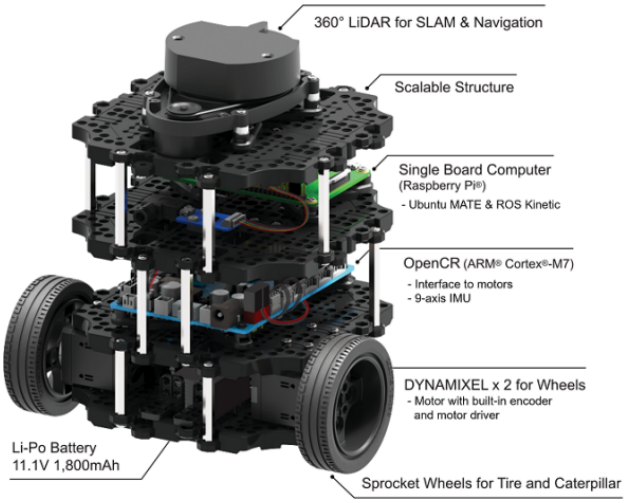


Figure 7. Configuration of mobile robot [2].

An interface board connects the left and right servomotors to a single board computer (SBC). SBC's operating system is Ubuntu Linux, and the middleware installed is ROS Kinetic [6]. The SBC is responsible for running the motor interface software module, which is depicted as the lowest layer in Fig. 6.

3.2. Robot Arm Inverse Kinematics

In this section, we create a software that calculates wheel rotational velocity and uses inverse kinematics to get the desired value of the robot velocity. In addition, we set up a teleoperation system for the robot by merging the program with another piece of software that creates the necessary robot speed based on joystick input.

To implement this, we first derive the inverse kinematics relation between the robot's velocity and its wheels' rotational velocity by manipulating equations 1 and 2 as follows:

$$v_r = v + \frac{T}{2}\omega \tag{8}$$

$$v_l = v - \frac{T}{2}\omega \tag{9}$$

$$w_r = \frac{v_r}{r} \tag{10}$$

$$w_l = \frac{v_l}{r} \tag{11}$$

After that, we fill in the blank lines inside `j2_inverse_kinematics_node.cpp` file, which declares the variables defined above. Lastly, we run the

program on the simulator and the actual robot.

There was no quantitative measurement for this experiment since the robot's position had not been specified. However, the robot was performing properly in accordance to the movement of the joystick, both on the simulator and the real test.

3.3. Kinematics and Odometry

In this task, we create software that uses kinematics to derive robot velocity from actual wheel speed. We also develop a program that computes robot position and orientation by combining robot velocity and odometry calculations. Finally, we build a pose estimation system by merging the program with another software that displays the estimated pose.

This task was done in the same way as the previous task: we fill in the blank inside script `j2_kinematics_node.cpp` and `j2_odometry_node.cpp` with physical quantity defined in equations 1, 2, 3, 4, 5, 10, and 11. Then, we employ the script on both the simulator and the actual robot.

First, we experimented with the robot's linear distance by moving the robot back and forth. Then, we measure both the traveled distance and displacement shown in the simulator. To minimize error due to the rotational movement of the robot (shifting the joystick analog to the left or right), we comment out all the commands that contain variable ω in the related scripts. The result obtained is shown in Table 1. It can be observed that there are errors between the real and simulated distance for both forward and backward movements. Furthermore, it can be highlighted that forward movement yields a bigger error than backward movement.

Second, the robot was tested to rotate both clockwise and counterclockwise to measure the angle accuracy. Similar to the linear movement test, we comment out all the commands that contain variable v in the related scripts to deaden the linear motion that might contribute error to the result. The result obtained is shown in Table 2. From this, we can observe that the clockwise movement gives a bigger error than the counterclockwise movement.

Several factors might contribute to the error yielded in the experiments mentioned above: a) there might be a slip between the wheels and the ground. As mentioned in Section 2.2, the odometry mechanism is prone to an error caused by slip. Moreover, this slip might be caused by dusty floors (or wheels), loose mats (we use mats as the ground), or the walking surfaces do not have same degree of traction in all areas [7]; b) The contact point between the wheels and the ground is shifted, which might increase or decrease the value of variable T shown in Figure 2. This might happen because of the slip or non-uniform mass distribution of the robot. The change of T might change the calculated pose as it is related to other variables (equations 1 and 2); c) the integral equation is simplified in the code by implementing repetitive summation with a small increment of dt. Of course, summation does not generate the same accuracy as integral; d) Since the measurement was done manually with human eyes and a ruler, human error also plays a role in contributing to the error.

3.4. Path Following

We modify the gains of the path following control and navigate the robot along a given path in this portion of the experiment. We also experiment with different gains and examine how the robot moves. The scheme of the mechanism of this experiment is depicted below.

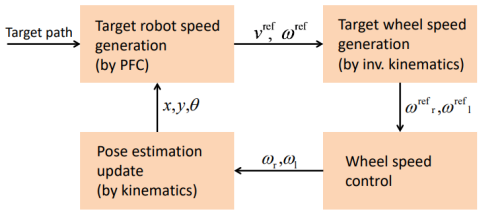


Figure 8. Path following control scheme [3].

For the implementation, we only change the variables `k_phi` and `k_d` inside the `line_following.xml` file. Then, we open another terminal: the first terminal opens the simulator, while the second runs the `.xml` file and generates the path.

The value of coefficients k_ϕ and k_d were defined in Equation 7. These coefficients were subject to change with a range of $0 \leq k_\phi \leq 1$ and $0 \leq k_d \leq 10$. Furthermore, during the experiment, we did not a employ specific value of these variables or design any particular path. Rather, we experimented with how each variable affects the robot's behavior by assigning random values (one value is fixed, while the other is set to a very high or very small number and vice versa).

When we set these coefficients with small values, the robot seems disoriented as sometimes the robot unnecessarily rotates a little during the linear motion. However, when the values are set to their respective maximums, the robot moves more straightforwardly and spends less time reaching the same endpoint compared to the previous case.

Based on our discussion and my understanding, these coefficients play a crucial role in determining the robot's behavior. By inspecting Equation 7 and Figure 4, we can deduce that when d and ϕ are large (say, the robot is moving from one corner to the other adjacent corner in a square path), the value of the reference angular acceleration is a large negative number. Assuming that the robot always possesses an angular velocity, the numerous decelerations will reduce the robot's angular velocity to a minuscule value where our eye

	Distance travelled (m)	
Movement	**Real**	**Simulation**
Forward	1	1.05
	0.5	0.526
Backward	1	1.01
Back and Forth	2	1.993

Table 1. Real vs. Simulated distance traveled by the robot. Note that the simulation distance obtained was calculated by taking mean of three measurements.

	Angle travelled (rad)	
Movement	**Real**	**Simulation**
Counterclockwise	6.28	6.43
	12.56	12.91
Clockwise	6.28	6.59
	12.56	13.17
Back and Forth (Clockwise first)	12.56	12.97
Back and Forth (Counterclockwise First)	12.56	12.44

Table 2. Real vs. Simulated angle traveled by the robot.

can no longer observe the rotating movement. Thus, if we set huge values of these coefficients, the aforementioned phenomenon will occur, where the robot focuses its movement to reach the next intermediate point first rather than thinking about its rotational movement (which explains the straightforward motion). On the other hand, if we set little values, the robot will become disoriented as it is confused to put more emphasis on rotational or linear motion (which explains the disorientation mentioned earlier). Finally, we can conclude that these coefficients give the robot instruction to which variable to give more attention to ($k_d \rightarrow d$, $k_\phi \rightarrow \phi$).

3.5. Costmap and Path Planning

In this task, we create software that will establish expenses on a known map. Furthermore, we examine alternative cost-cutting policies and direct the robot along (1) the shortest path, (2) a path as far away from walls as feasible (safest path), and (3) a path that passes via a corridor rather than rooms. These scenarios are depicted in Figure 9.

The implementation was done by changing the cost setter function, which its types are described below, inside the `j2_layer.cpp` script.

- `updatePoint(costmap, x, y, new_cost)`: Update the cost of point (x, y) with `new_cost`.
- `updateRectangle(costmap, x1, y1, x2, y2, new_cost)`: Update the cost of rectangle whose corners are (x_1, y_1) and (x_2, y_2) with `new_cost`.
- `updateCircle(costmap, x, y, r, new_cost)`: Update the cost of circle with center at (x, y) and radius r with `new_cost`.
- `updateNearObstacle(costmap, r, new_cost)`: Update the cost of an area within radius r from obstacle's coordinate with `new_cost`.

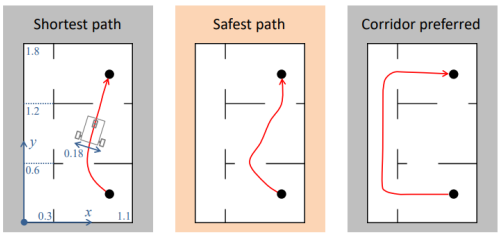


Figure 9. Path planning for different scenarios [3].

Moreover, we also transform the `new_cost` argument of the cost-setter function to determine how much cost is needed to pass through the specified area. There are several kinds of cost value:

- `OBSTACLE_COST()`: Infinite cost for a grid occupied by an obstacle.

- `COLLISION_COST()`: Infinite cost for a grid on which collision with an obstacle must happen.
- `FREE_SPACE_COST(difficulty)`: Positive cost for a traversable grid. Difficulty can be set $0 \sim 1$.

Thus, we can create the desired path by adjusting the cost-setter policies and their corresponding cost values.

For the shortest path scenario, we utilized the cost setter function `updateNearObstacle` with a radius of 0.1 and cost value as `COLLISION_COST()`, since we have to consider the robot's radius. However, we do not want to keep the robot too far from the obstacle as it will take a larger distance. For the safest path, we use the same command as the shortest path but with a different radius of $r = 0.13$. With a larger radius, the robot is guaranteed not to hit and possesses a safe distance from the obstacle. Lastly, we employ the same command as the safest path with an additional `updateRectangle` function with `COLLISION_COST()` cost value for the corridor preferred path. The corners of the rectangle are set to be $(0.3, 0.6)$ and $(1.1, 1.2)$. By implementing these two cost-setter functions, the robot cannot pass through the middle of the map and is obliged to pass through the corridor with a guarantee of not hitting the obstacle (safest path scenario).

4. Assignments

4.1. Assignment 1

Discussion: *Because the robot pose estimation by the kinematics and odometry assumes perfect grip between the wheels and ground, it is likely to produce estimation error when slip happens. To solve this problem, propose (1) slip estimation method, or (2) alternative position estimation method. Also, for (1), introduce the type of sensor(s) you use and its model number, and discuss the accuracy of slip estimation expected from the sensor specs. For (2), introduce the type of sensor(s) you use and details of position estimation method.*

For the slip estimation method, one relevant paper discusses a different type of slippage and its model for skid-steering tracked vehicles [8]. Firstly, the type of slippage introduced in this paper are:

- Longitudinal slippage during straight maneuvers: Longitudinal slippage is estimated based on the terramechanics force interaction model between the track and the terrain. The authors propose a simplified slippage model that uses approximation to reduce terrain-dependent parameters
- Lateral slippage during straight maneuvers: Lateral slippage is also estimated based on the terramechanics force interaction model between the track and the terrain. The authors propose an approximation slippage model with one terrain and robot dependent parameter.
- Longitudinal slippage during turning maneuvers: Skid-steering slippage is a complex phenomenon, and it requires many parameters for estimating the terramechanics-based slippage. As mentioned above, the empirical slip estimation formula [9] was confirmed for rigid fat ground. The authors verify whether the formula can be applied to loose and weak slopes.
- Lateral slippage during turning maneuvers: To estimate lateral slippage while a robot is turning, regression analysis has been used along with training data obtained from the environment. The training data include inertial information and robot position.

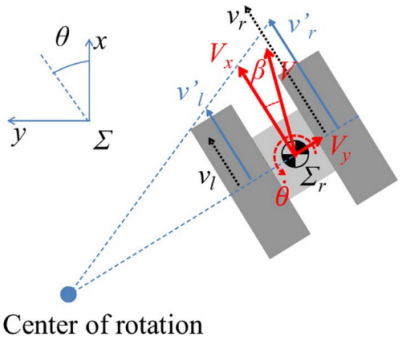


Figure 10. Kinematics of a skid-steering tracked vehicle [8].

The authors also propose slip-compensated odometry to apply the slip model to the kinematics of skid-steering vehicles (scenario used in the experiment). The kinematics of a skid-steering mobile robot, including its longitudinal and lateral slippage is depicted in Figure 10. When the right and left wheels rotate respectively at v_r and v_l, the longitudinal and lateral translational velocities V_x and V_y in robot coordinate system Σ_r are expressed as follows:

$$V_x = \frac{v_r(1-\alpha_r) + v_l(1-\alpha_l}{)}2 \tag{12}$$

$$V_y = V_x \cdot \tan\beta \tag{13}$$

where α_r and α_l represent slip ratios corresponding to the two wheels, and β is the slip angle. The slip ratio and slip angle represent the longitudinal and lateral slippage, respectively. The slip ratios of the wheels are defined as follows:

$$\alpha_r = 1 - \frac{v_r'}{v_r} \tag{14}$$

$$\alpha_l = 1 - \frac{v_l'}{v_l} \tag{15}$$

where v_r' and v_l' represent ground velocities, and v_r and v_l are input velocities that can be measured by encoders. Notice that if the robot runs without longitudinal slippage, the ground velocity equals the input velocity, and the slip ratio is zero. Slip angle β is the angle of deviation between the desired transition direction and the actual direction of motion of the robot. In Fig. 10, V_x and V_y are ground velocity components of the robot base. The kinematics of the tracked vehicle in coordinate system Σ can, therefore, be derived as follows:

$$\dot{x} = V_x \cos\theta - V_y \sin\theta \tag{16}$$

$$\dot{y} = V_x \sin\theta + V_y \cos\theta \tag{17}$$

$$\dot{\theta} = \frac{v_r(1-\alpha_r) + v_l(1-\alpha_l)}{T} \tag{18}$$

where θ and T were already mentioned in Section 2.2. Slip-compensated odometry is obtained by integrating from the kinematics expressed in Eqs. 16-18.

After several assumptions and calculations, the slip model for each direction and maneuver is obtained and shown in Table 3. Here, R_c is the compaction resistance, W is the weight of the wheel, θ_{pitch} is pitch angle, θ_{roll} is roll angle, C and C' are constants, n is a robot-dependent empirical slip parameter, X_i is the i-th explanatory variable, and a_i is the i-th coefficient of the variable (note that X_i and a_i are used for regression).

To measure the ground truths corresponding to the robot's movement and orientation, Osprey motion-capture system is installed in the corner of the experimental field. The obtained result is shown in Figure 11, which compares the estimated position results with the ground truth, gyrodometry result, and result obtained from conventional method.

It can be seen that the proposed model is able to achieve high accurate result compared to the other methods. Moreover, we can deduce that gyrodometry does not consider slippage, which results in longer position. Likewise, the conventional method only considers longitudinal slippage, and the lateral slippage causes an increase in the position estimation error.

4.2. Assignment 2

Discussion: *Compare kinematics (forward and inverse) of the differential drive type robot and robot arm (manipulator), and emphasize their difference.*

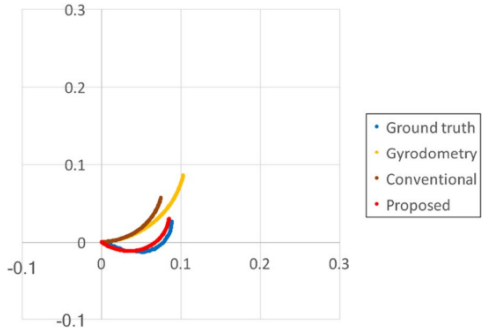
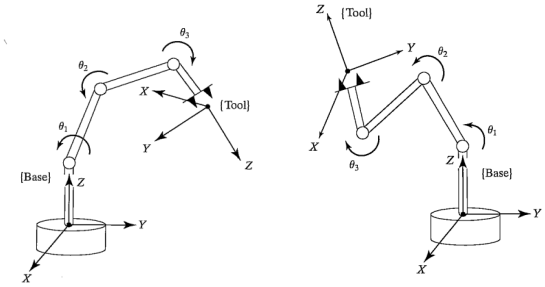


Figure 11. Result of position estimation: slope angle is $15°$; input velocity is $5\ cm/s$, and input angular velocity is $30°/s$ [8].

In my opinion, the most prominent difference between the differential drive type robot and the robot arm is the implementation of the forward and inverse kinematics. As discussed in Section 2.2, kinematics determines the robot's velocity based on the angular velocity of the wheels, while inverse kinematics does the opposite thing. In the robot arm, The forward kinematic problem is to compute the position and orientation of the tool frame by describing the tool frame, which is attached to the end-effector, relative to the base frame, which is attached to the nonmoving base of the manipulator ($(\theta_1, \theta_2, \theta_3) \rightarrow (X, Y, Z)$, see Figure 12(a)). On the other hand, inverse kinematics for arm manipulator is the case where given the position and orientation of the manipulator's end-effector, calculate all possible joint angles that could be used to attain the end-effector's position and orientation ($(X, Y, Z) \rightarrow (\theta_1, \theta_2, \theta_3)$, see Figure 12(b))[10].



(a) Forward kinematics (b) Inverse kinematics

Figure 12. Robot arm manipulator forward and inverse kinematics [10].

Another difference between both cases is the occurrence of singularity. In differential drive type robot, the singularity limits the robot's movement in the direction along the common axis of the drive wheels [1]. On the other hand, singularity in robotic arm is more complex [11]:

- Boundary singularities: Caused by a full extension of a joint, and asking the manipulator to move beyond

	Slip ratio	Slip angle
Straight Maneuver	$\alpha = \frac{R_c}{C} + \frac{W}{C}\theta_{pitch}$	$\beta = \frac{W}{C'}\theta_{roll}$
Turning Maneuver	$\frac{\alpha_l}{\alpha_r} = -sgn(v_r \cdot v_l)\left(\left\|\frac{v_r}{v_l}\right\|\right)^n$	$\beta = \Sigma_{i=0}^{7} a_i X_i$

Table 3. Slip models for each direction and maneuver

where it can be positioned. Typically, this is trying to reach out of the workspace at the farthest extent of the workspace.

- Internal singularities: Caused by an alignment of the robots axes in space. For instance, if 2 axes become aligned in space, rotation of one can be canceled by counter rotation of the other, leaving the actual joint location indeterminate. Also, certain kinematic alignments specific to each manipulator can cause these.

The singular positions of the robot arm can be found by analyzing the Jacobian matrix of the manipulator.

Lastly, the computation of kinematics of the robot arm is relatively much more difficult than the drive type robot. Using equation 1 and 2, we can obtain the formulas used in kinematics and its inverse. On the other hand, depending on the number of joints, the calculation complexity of the robot arm mechanism might increase to an extent where it cannot be done by hand. For instance, if there are 5 links, we have to calculate the inverse matrix of 5×5 matrix for its inverse kinematics.

4.3. Assignment 3

Discussion: *Compare kinematics (forward and inverse) of the differential drive type robot and robot arm (manipulator), and emphasize their difference.*

The differential drive is a two-wheeled drive system with independently controlled actuators for each wheel. The term refers to the fact that the robot's motion vector is the total of the separate wheel motions, which is also true of the mechanical differential (however, this drive system does not use a mechanical differential). As indicated in Figure 1, the drive wheels are typically located on each side of the robot (either in the front or rear).

In differential wheel type, straight-line motion is achieved by moving the driving wheels in the same direction at the same velocity. In-place (zero turning radius) rotation is accomplished by turning the driving wheels in the opposite direction at the same velocity. By dynamically altering the angular velocity and/or direction of the driving wheels, arbitrary motion routes can be constructed. In reality, however, motion routes are implemented as alternating sequences of straight-line translations and in-place rotations, which reduces complexity [12].

Differential wheel is advantageous as it is simple (which is why it is usually applied in robot). However, it also comes with a drawback: It is hard to make a differential robot moves in a straight line. Because the driving wheels are independent, if they are not turning at the same rate, the robot will veer to one side. Due to minor changes in the motors, friction differences in the drive trains, and friction variances at the wheel-ground interface, making the drive motors turn at the same pace is difficult. Solving this problem may necessitate the use of interrupt-based software and assembly language programming. It is also critical to obtain precise wheel position information, which usually comes from odometry.

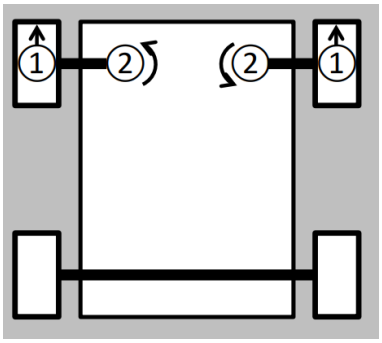


Figure 13. Car-type drive [3].

A separate set of steering wheels and a pair of driving wheels are characteristics of car-like movement (see Figure 13). A car-type drive is one of the simplest locomotion systems, with separate motors controlling translation and turning (a significant advantage compared to the differential drive system). This is why it is well-known for its human-driven cars. The simplicity, however, comes at a cost: the turning mechanism must be properly regulated. Odometry inaccuracies can be caused by minor position errors in the turning mechanism. Path planning is also difficult if the environment necessitates motion in a direction where there is no direct actuation (sideways). Because no actuator can immediately perform a sideways movement, a car must undertake numerous forward/reverse motions in parallel parking to achieve a sideways translation [13].

4.4. Assignment 4

Discussion: *The differential drive mechanism can be considered as "non-holonomic" mechanism. On the other hand, there is "holonomic" mechanism. Introduce their definition and difference.*

The term Holonomic refers to the relationship between a robot's controllable and total degrees of freedom. The robot is said to be Holonomic if the controlled degree of freedom equals the total degree of freedom. A robot based on castor wheels or Omni wheels (see Figure 14) is an excellent demonstration of Holonomic drive since it can move freely in any direction and its controllable degrees of freedom equals its total degrees of freedom. The illustration depicts a castor wheel with the ability to rotate on both the x and y axes, allowing it to move in both directions [14].

Figure 14. The moving mechanisms of an omni-directional wheel [15].

A non-Holonomic drive occurs when the controlled degree of freedom is less than the total degree of freedom. A automobile has three degrees of freedom: position in two dimensions and orientation. However, there are only two degrees of freedom that may be controlled: acceleration (or braking) and steering wheel turning angle. This makes turning the car in any direction difficult for the driver (unless the car skids or slides) [14].

5. Conclusion

This experiment taught us to simulate a differential drive type robot using a program and a joystick. We also investigate the kinematics and odometry error by measuring the robot's virtual and real distance traveled. Finally, we utilize the cost-setter function and its corresponding cost value to design the shortest path, safest path, and corridor preferred path.

References

[1] Gregory Dudek and Michael Jenkin. *Computational Principles of Mobile Robotics*. 2nd ed. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511780929 (see pp. 1, 7).

[2] Shotaro Kojima. "Control methods for differential drive type mobile robot - Textbook". In: 2021. URL: https://drive.google.com/drive/folders/1Wbn2WQt4TdHwZVb3lc4thIzSbHHKeLGf (see pp. 1, 3).

[3] Shotaro Kojima. "Control methods for differential drive type mobile robot - Slides". In: 2021. URL: https://drive.google.com/drive/folders/1Wbn2WQt4TdHwZVb3lc4thIzSbHHKeLGf (see pp. 1, 2, 4, 5, 8).

[4] S. Iida and S. Yuta. "Vehicle command system and trajectory control for autonomous mobile robots". In: *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*. 1991, 212–217 vol.1. DOI: 10.1109/IROS.1991.174452 (see p. 2).

[5] *TurtleBot3*. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (see p. 3).

[6] *ROS Documentation*. URL: http://wiki.ros.org/ (see p. 3).

[7] Canadian Centre for Occupational Health Safety. *Prevention of Slips, Trips and Falls*. URL: https://www.ccohs.ca/oshanswers/safety_haz/falls.html (see p. 4).

[8] Genki Yamauchi et al. "Slip-compensated odometry for tracked vehicle on loose and weak slope". In: *ROBOMECH Journal* 4.1 (Nov. 2017), p. 27. ISSN: 2197-4225. DOI: 10.1186/s40648-017-0095-1. URL: https://doi.org/10.1186/s40648-017-0095-1 (see pp. 6, 7).

[9] Daisuke Endo et al. "Path following control for tracked vehicles based on slip-compensating odometry". In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2007, pp. 2871–2876. DOI: 10.1109/IROS.2007.4399228 (see p. 6).

[10] Tony Owen. "Introduction to Robotics: Mechanics and Control by John J. Craig Addison-Wesley Publishing Company, Massachusetts, USA, 1986, (£17.95, student hardback edition)." In: *Robotica* 6.2 (1988), pp. 164–165. DOI: 10.1017/S0263574700004045 (see p. 7).

[11] Columbia University. *Kinematics Singularities and Jacobians*. URL: http://www.cs.columbia.edu/~allen/F15/NOTES/jacobians.pdf (see p. 7).

[12] *Robo-Rats Locomotion: Differential Drive*. URL: https://groups.csail.mit.edu/drl/courses/cs54-2001s/diffdrive.html (see p. 8).

[13] *Robo-Rats Locomotion: Car-type Drive*. URL: https://groups.csail.mit.edu/drl/courses/cs54-2001s/car.html (see p. 8).

[14] Robot Platform. *Robot Locomotion*. URL: http://www.robotplatform.com/knowledge/Classification_of_Robots/Holonomic_and_Non-Holonomic_drive.html (see p. 9).

[15] Moein Jafari. *GET ROLLING WITH OMNI-DIRECTIONAL WHEELS*. URL: https://www.servomagazine.com/magazine/article/get-rolling-with-omni-directional-wheels (see p. 9).