0301-346: Project III
Robot Kinematics using Inheritance and Polymorphism

Fall 2020

I. Introduction

In this project, we will be implementing mobile robot kinematic equations using C++ and classes that represent two different mobile robot configurations. Kinematics can be used to obtain a mobile robot's trajectory based on wheel type, configuration, position, and velocity. For the scope of this course you will not be asked to derive the kinematic equations, but you are expected to be able to use them.

In this project the student will create a class hierarchy that includes an abstract class called *Robot*, a derived abstract class *MobileRobot* and two concrete derived classes *DifferentialDriveRobot* and *OmnidirectionalRobot*. These classes will be used create robot objects that may be dynamically created and modified during execution of the program. Additionally, a class *Wheel* will be created to represent wheels that belong to a *MobileRobot*. *DifferentialDriveRobot* and *OmnidirectionalRobot* will implement pure virtual functions defined in *MobileRobot* that will use forward kinematic equations to calculate linear and rotational velocities of the two different robot configurations based on their wheels' velocities. Lastly, a print function will be implemented for each object and will be implemented at each level of the hierarchy, where each derived class will call its base class's print function.

Through the completion of this project, the student will gain knowledge in creating a class inheritance hierarchy, the differences with *is-a* and *has-a* relationships, how to use pointer handles, down-casting, polymorphism using virtual/pure virtual functions, and abstract classes.

II. Mobile Robot Kinematics

Kinematics, as they relate to wheeled mobile robots, defines the relationship between the controlled motors (actuators) and robot's resulting position in its environment. If the wheels' configuration and velocities are known, then the robot's linear velocity and rotational velocity may be solved for. This process is known as *forward kinematics*. Using the linear and rotational velocities found using forward kinematic equations, with the addition of time, the final position of the mobile robot in its environment may be determined. However, if the final position of the robot is the desired input, then *inverse kinematics* is used solve for the wheels' velocities. Kinematic equations are different based on robot configuration and must be solved on a per robot basis. For the scope of this project, the general robot configuration and its corresponding kinematic equations will be given.

III. *Differential Drive Robot*

A differential drive robot, as it pertains to this specific project, refers to a mobile robot that consists to two drive wheels located along the robot's central *y*-axis. These two drive wheels may be independently driven, and no other balancing wheels/castors are considered. The robot is assumed to balance perfectly upon its drive wheels. The configuration layout of the differential drive robot is shown in Figure 1.

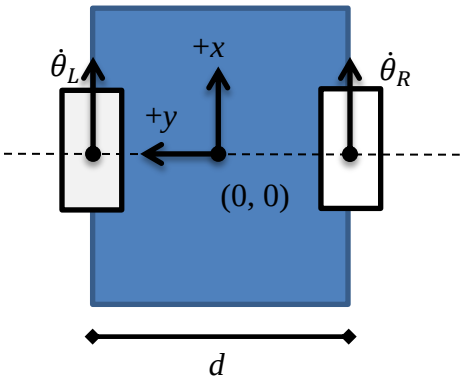

Figure 1 – Two-wheeled differential drive robot configuration.

From Figure 1, the only variables for the configuration of the differential drive robot are the distance between the drive wheels, *d*, and the radius of the drive wheels, *r* (not shown). Using forward kinematics, the linear velocity, *v*, and rotational velocity, *ω*, can be found using Equation 1 and Equation 2 or by using matrices in Equation 3. The definitions of the variables and their respective units are listed in Table 1.

$$v = \pi r\left(\dot{\theta}_L + \dot{\theta}_R\right) \tag{1}$$

$$\omega = \frac{2\pi r}{d}\left(\dot{\theta}_R - \dot{\theta}_L\right) \tag{2}$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = 2\pi r \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{d} & \frac{1}{d} \end{bmatrix} \begin{bmatrix} \dot{\theta}_L \\ \dot{\theta}_R \end{bmatrix} \tag{3}$$

Once, the linear velocity and rotational velocity of the robot is calculated the respective x and y components of the linear velocity may also be found, using Equation 4 and Equation 5.

$$v_x = v \cdot \cos(\omega) \tag{4}$$

$$v_y = v \cdot \sin(\omega) \tag{5}$$

Table 1- Variables for differential drive robot.

Variable	***Description***	***Units***
v	Robot linear velocity	m/s
ω	Robot rotational velocity	rad/s
$\dot{\theta}_L$	Left wheel velocity	rev/sec
$\dot{\theta}_R$	Right wheel velocity	rev/sec
r	Wheel radius	m
d	Distance between wheels	m
v_x	Robot linear velocity in x direction	m/s
v_y	Robot linear velocity in y direction	m/s

Visual representation of robot movement for a differential drive robot based on different wheel velocities is shown in Figure 2.

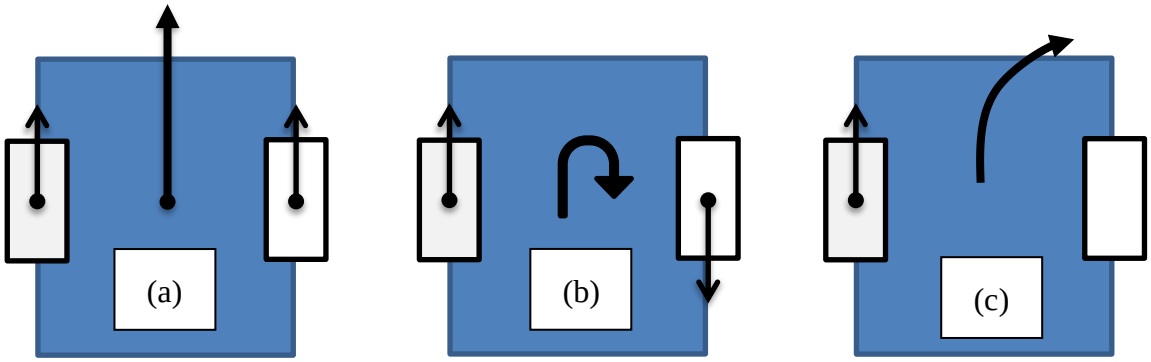

Figure 2 - Direction of differential drive robot based on wheel velocities for (a) forward motion (b) clockwise rotation about center and (c) curve to the right.

IV. Omnidirectional Drive Robot

Omnidirectional drive robots refer to a class of mobile robots that are capable of driving in all directions. The mechanical process behind the ability of an omnidirectional drive robot to drive in all directions is in its wheels. The most common wheel configurations for an omnidirectional drive robot are Mecanum wheels. Mecanum wheels are special wheels where their surface is covered with free rolling cylinders angled at either 45° or 90°. Although the rollers are freely rotating, they are only freely rotating when placed parallel to the wheel axis. This concept allows for movement in parallel and perpendicular directions with respect to the drive wheels.

For this project, a four wheeled omnidirectional robot configuration will be used as seen in Figure 3. Each of the four wheels may be independently driven and are 45° Mecanum

wheels. The rollers in the wheels on the left side are at +45° with respect to the wheel axis and the rollers in the right side wheels are at -45°.

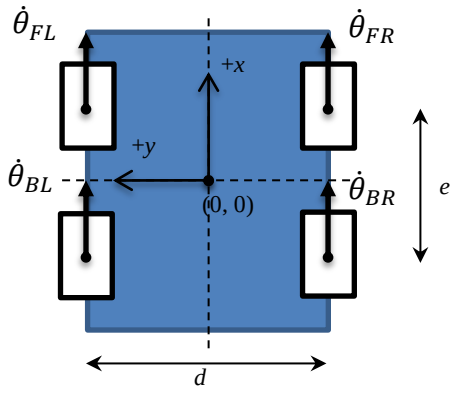

Figure 3 - Four-wheeled omnidirectional drive robot configuration.

From Figure 3, the only variables for the configuration of the omnidirectional drive robot are the distance between the drive wheels, *d*, the distance between the front and back wheels, *e*, and the radius of the drive wheels, *r* (not shown). Using forward kinematics, the linear velocities of the robot in the *x* and *y* directions may be calculated using Equation 6 and Equation 7. Additionally, the rotational velocity of the may be found using Equation 8. Alternately, linear velocities and rotational velocities may simultaneously found using the matrix-form in Equation 9. The definitions of the variables and their respective units are listed in Table 2.

$$v_x = \frac{\pi r}{2}\left(\dot{\theta}_{FL} + \dot{\theta}_{FR} + \dot{\theta}_{BL} + \dot{\theta}_{BR}\right) \tag{6}$$

$$v_y = \frac{\pi r}{2}\left(-\dot{\theta}_{FL} + \dot{\theta}_{FR} + \dot{\theta}_{BL} - \dot{\theta}_{BR}\right) \tag{7}$$

$$\omega = \frac{\pi r}{(d+e)}\left(-\dot{\theta}_{FL} + \dot{\theta}_{FR} - \dot{\theta}_{BL} + \dot{\theta}_{BR}\right) \tag{8}$$

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = 2\pi r \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{2(d+e)} & \frac{1}{2(d+e)} & -\frac{1}{2(d+e)} & \frac{1}{2(d+e)} \end{bmatrix} \begin{bmatrix} \dot{\theta}_{FL} \\ \dot{\theta}_{FR} \\ \dot{\theta}_{BL} \\ \dot{\theta}_{BR} \end{bmatrix} \tag{9}$$

Table 2 - Variable definitions and units for omnidirectional drive robot.

Variable	***Description***	***Units***
v	Robot linear velocity	m/s
ω	Robot rotational velocity	rad/s
$\dot{\theta}_{FL}$	Front-Left wheel velocity	rev/sec
$\dot{\theta}_{FR}$	Front-Right wheel velocity	rev/sec
$\dot{\theta}_{BL}$	Back-Left wheel velocity	rev/sec
$\dot{\theta}_{BR}$	Back-Right wheel velocity	rev/sec
r	Wheel radius	m
d	Distance between wheels	m
e	Distance between front and back wheels.	m
v_x	Robot linear velocity in *x* direction	m/s
v_y	Robot linear velocity in *y* direction	m/s

Once the linear velocities in the *x* and *y* direction are found, the magnitude of the linear velocity of the robot can be calculated using Equation 10.

$$|v| = \sqrt{{v_x}^2 + {v_y}^2} \tag{10}$$

The direction of the linear velocity can be set using the angle found by the arctangent function. C++ and many other languages provide libraries that implement a special form of the arctangent functions called *atan2*, which takes the *x* component and *y* component, then return the resulting vector angle corrected for the proper quadrant in 2D Euclidean space. Equation 11 can be used to compute the angle of the linear velocity vector, φ of the robot based on the orientation of axis on the robot.

$$\varphi = atan2(v_x, -v_y) \tag{11}$$

From the angle found in Equation 11, the direction of velocity vector may be determined based on the quadrant of the resultant vector as formulated in Equation 12.

$$v = \begin{matrix} -|v| \\ |v| \end{matrix} \quad \begin{matrix} if\ [(\varphi > \pi) and (\varphi < 2\pi)]\ or\ [(\varphi < 0) and (\varphi > -\pi)] \\ otherwise \end{matrix} \tag{12}$$

Visual representation of robot movement for an omnidirectional drive robot based on different wheel velocities is shown in Figure 4.

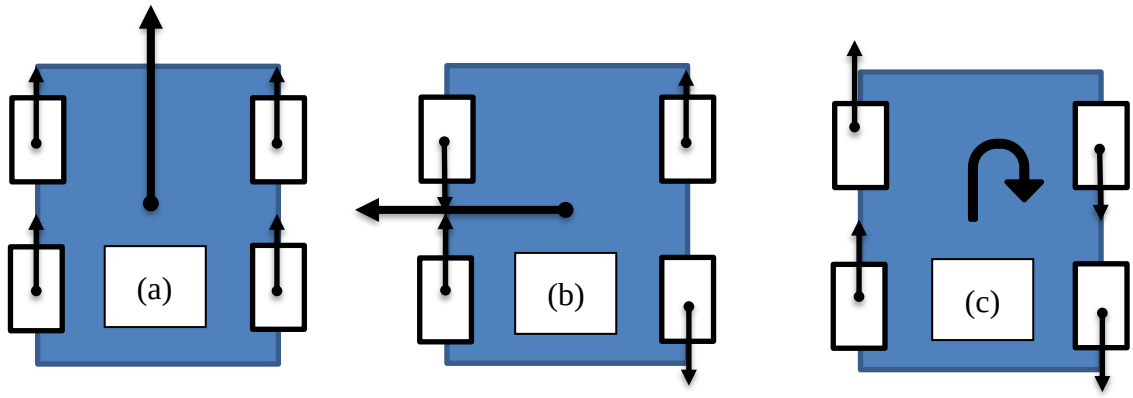

Figure 4 - Direction of omnidirectional robot based on wheel velocities for (a) forward motion (b) sliding left motion and (c) clockwise rotation about center.

V. Class Hierarchy

For this project you will implement a class hierarchy as shown in Figure 5. Figure 5 is an UML (Unified Modeling Language) diagram that specifies classes, class members, class member functions, access properties, and relationships. The UML diagram was created in Visio and as a note, "-" means private, "+" public, members are on the top half, member functions on the bottom half, data types are specified after a ":", and "in" means input parameter/argument.

Additional Notes on Class Hierarchy

- *Robot* is an abstract class
    - Its *print* member function is a pure virtual and not implemented at the *Robot* scope, thus a user cannot create an instance of *Robot*
    - *model* is a protected member variable, can be directly accessed in any derived class of *Robot*
    - Has **no** default constructor, derived class must call *Robot* constructor in member initializer list.

- *MobileRobot* is an abstract class
    - Member functions *getVel*, *getRotVel*, *getVx*, *getVy*, are pure virtual functions and are not implemented at the *MobileRobot* scope, thus the user cannot create an instance of *MobileRobot*.
    - Member function *print* is a pure virtual function, but it is implemented at the *MobileRobot* scope.
    - Has **no** default constructor, derived class must call *MobileRobot* constructor in member initializer list.
- *DifferentialDriveRobot* and *OmnidirectionalDriveRobot* are concrete classes.
    - They must implement functions *getVel*, *getRotVel*, *getVx*, *getVy*, and *print*.

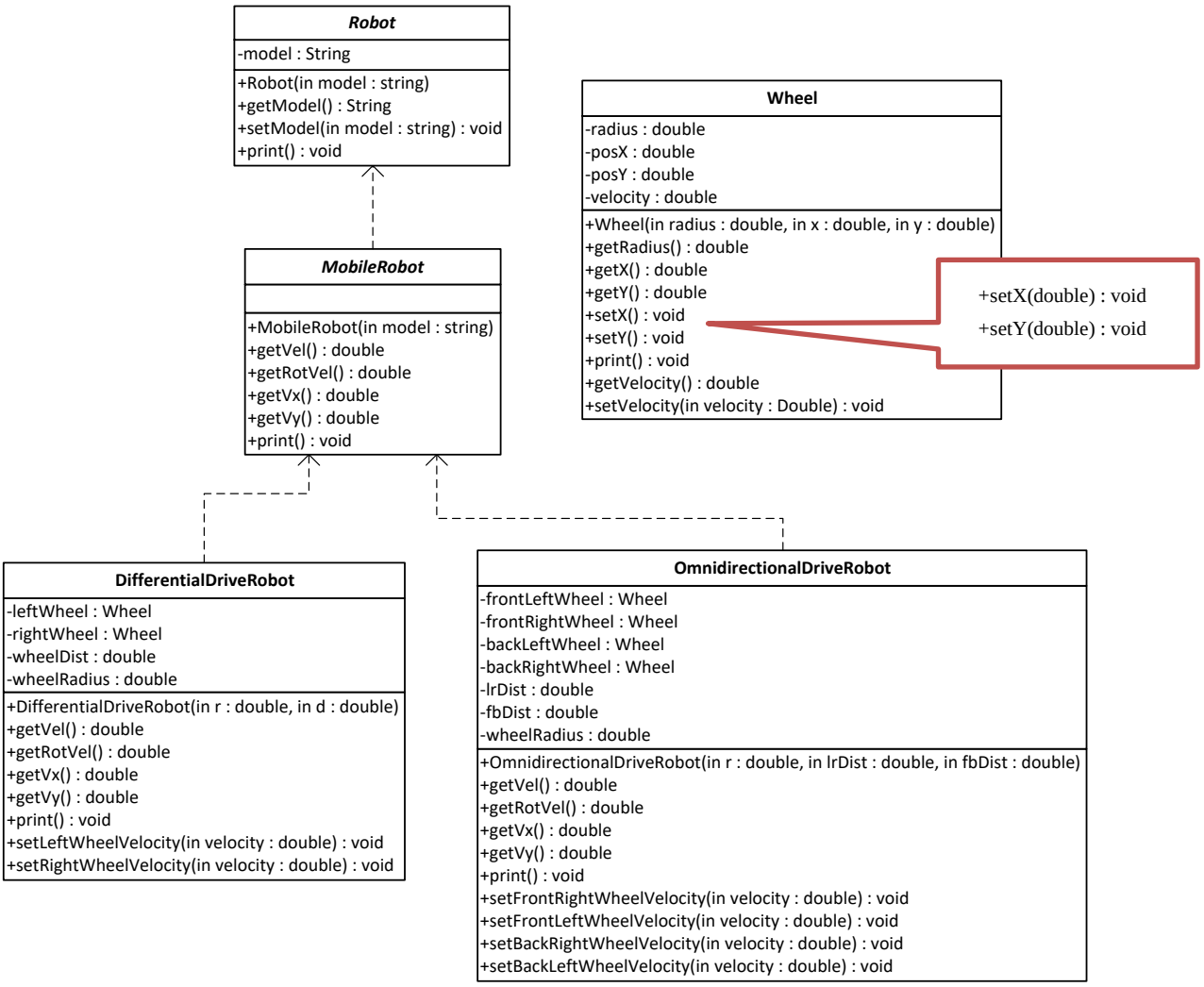

Figure 5 - Class hierarchy for mobile robots.

VI. **[50pts] Project Requirements**

1. [2pts] Define/Implement *Robot* Class
    i. Only constructor needs implemented

2. [3pts] Define/Implement *MobileRobot* Class
    i. Implement constructor and print member function

3. [5pts] Define/Implement *Wheel* Class
    i. For print function use cout and display:

```
Position: (x,y)
Velocity: xxxx rev/sec
```

4. [15pts] Define/Implement *DifferentialDriveRobot* Class
    i. [5pts] getVel
    ii. [5pts] getRotVel
    iii. [2pts] getVx, getVy
    iv. [3pts] print
        1. Call print functions for the wheels
        2. Output v, ω, v_x, v_y

5. [25pts] Define/Implement *OmnidirectionalDriveRobot* Class
    i. [5pts] getVx
    ii. [5pts] getVy
    iii. [5pts] getRotVel
    iv. [5pts] getVel
    v. [5pts] print
        1. Call print functions for the wheels
        2. Output v, ω, v_x, v_y

VII. **[40pts] Program Execution (main.cpp)**

1. [5pts] Create a pointer handle of *MobileRobot* type called robot
    i. DO NOT create any additional pointer handles, must reuse.

2. Prompt user which Robot to create, Differential Drive or Omnidirectional Drive
    i. [10 pts] Differential Drive selected
        1. Prompt user for wheel radius and distance between wheels
        2. Point robot to a *new DifferentialDriveRobot* object.
        3. ~~Call print function,~~ ~~**must not use down-casting**~~
        4. Prompt user for wheel velocities
        5. Set wheel velocities, **must use down-casting**
        6. Call print function
        7. Repeat 4-6 till user selects not to.
    ii. [20 pts] Omnidirectional Drive selected
        1. Prompt user for wheel radius, distance between wheels (left-right and front-back).
        2. Point robot to a *new OmnidirectionalDriveRobot* object.
        3. ~~Call print function,~~ ~~**must not use down-casting**~~
        4. Prompt user for wheel velocities
        5. Set wheel velocities, **must use down-casting**
        6. Call print function
        7. Repeat 4-6 till user selects not to.

3. [3 pts] Repeat step 2 until user selects not to.
4. [2 pts] Delete pointer.

VIII. **[10pts] Program Submission**

1. Submit ONLY cpp and h files
2. Define class definitions in *.h files
3. Implement class functions in *.cpp files
4. Place main function in main.cpp
5. Files required and ONLY ones need:
    i. Robot.h, Robot.cpp
    ii. MobileRobot.h, MobileRobot.cpp
    iii. DifferentialDriveRobot.h, DifferentialDriveRobot.cpp
    iv. OmidirectionalDriveRobot.h, OmnidirectionalDriveRobot.cpp
    v. Wheel.h, Wheel.cpp
    vi. main.cpp