# Homework assignment 1: Differential Drive Kinematics

In this assignment we created a Robot class which saves the robot state and does all the required functionalities. The Robot class is initialized in the main(), just before MyFronEnd class, which receives the robot object as an argument.

## 1. Control the robot using the keyboard and send appropriate motor settings.

When we press the up and down arrows we are setting the linear velocity of the robot to a constant and when we press left or right we are setting the angular velocity to a constant.

When we release any button we set either the linear velocity (for the up and down arrows) or the angular velocity (for the left and right arrows) to zero.

To make sure we don't set the velocities too high, we always divide the velocity of each wheel with the maximum speed it could have. That way these velocities will never be greater than 100. To calculate the maximum speed we divided the steps per second at 100% by the steps per revolution multiplied by wheel radius and 2pi.

To calculate these left and right velocities, we used the the differential drive forward kinematic equations given in class.

As the velocities can't be negative, we take the absolute value of them and then we change the direction of the wheels with the flag right_dir or left_dir.

## 2. Maintain the robot pose by integrating over time.

In the MyFrontEnd class update() method we call robot.updateVariables(). That method calculates the updates of the new robot position and angle as following:

1. Calculate movement transformation. This matrix includes the new forward/backward movement and angular velocity. We take care of the desired numbers by multiplying by time_delta.
2. Calculate robot to map transformation
3. Dot product between robot to map transformation and movement transformation

4. Extract new cx, cy and angle from the resultant matrix

## 3. Draw the robot on the screen.

In the MyFrontEnd class draw() method we call robot.draw_robot(). That method draws the robot in the surface as following:

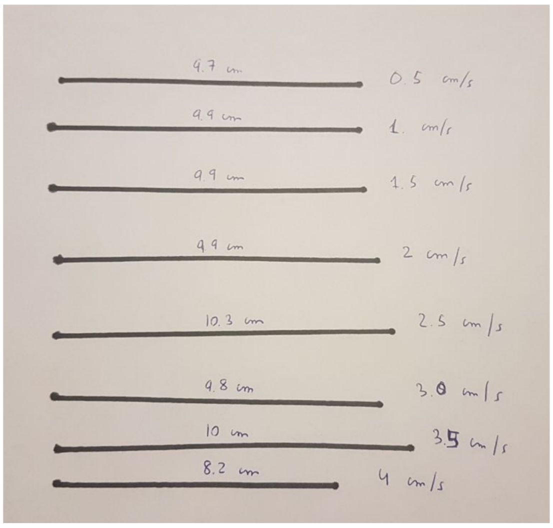
1. Calculate the four points corners coordinates of the robot in the map frame using a function called robot_to_map_frame(). This function does the dot product between the robot to map transformation and the point in the robot frame. Its output is the coordinates of the point in the map frame.
2. Calculate the points coordinates of the sonar in the map frame using a function called sonar_to_map_frame(). This function calculates the transformation matrix from sonar to robot using the servo_angle. Then it does the triple dot product between robot to map transformation, sonar to robot transformation and the point coordinates in the sonar frame. So we use this function to calculate the point coordinates of the starting point of the sonar and the point coordinates of the ending point of the sonar using the sonar distance of Sparki.
3. Draw all the lines using the points calculated before projecting them to a non-homogeneous coordinates.

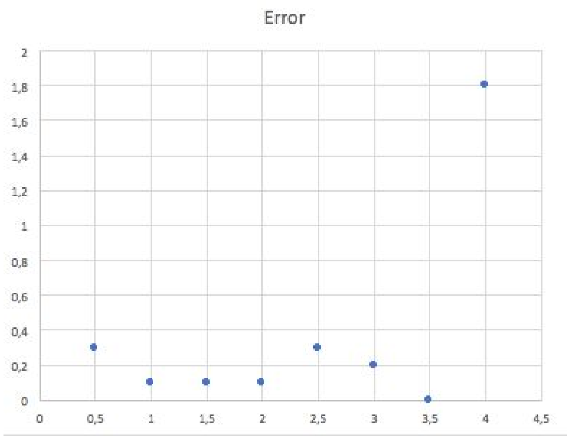
## 4. Evaluate your robot navigation system.

To evaluate our robot navigation system we have done the following experiments. Homework PDF says that we should use time.sleeps() but using that would interfere with the thread of the program and messages would be dropped. In order to handle this with a similar function we have created a wait() function which uses the class Timer in order to execute a function after some specific time.

### 4.1 Move the robot 10 cm in differing amounts of time

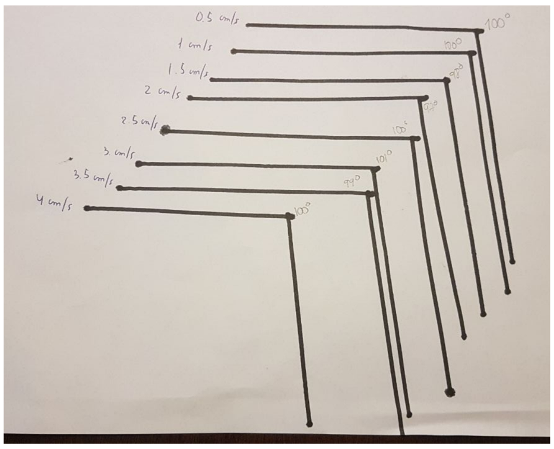
We measured the observed error in movement by measuring the line that Sparki drew with a ruler and then we made a plot of error vs. velocity.

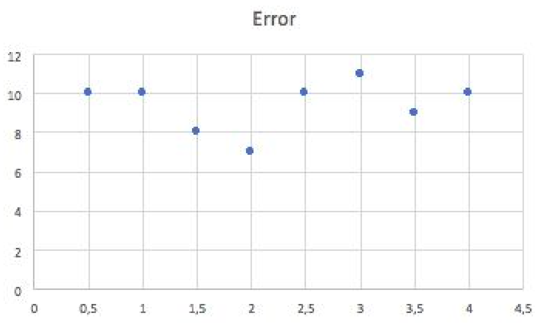




4.2 Move the robot forward 10cm, rotate the robot 90 degrees, and move forward 10cm again. Complete the rotation in differing amounts of time.

We measure the error in the turn by measuring the angle between the two lines using a protractor and then we made a plot of error vs. velocity.

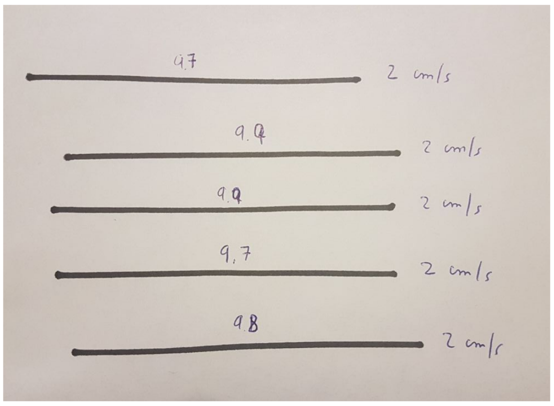




4.3 Analyze the results

When drawing a single line we were expecting 10 cm for each velocity, except for those that go higher than 3.8 cm/s, which is the maximum velocity our Sparki could go. We expected that after a 3.8 cm/s the lines would start to be smaller and smaller because the program thinks that the robot can go at certain velocity and it has a maximum. We can see this effect when going at 4 cm/s.

For the velocities below 3.8 cm/s we thought that the error could be caused for things like the grip of the wheels, friction between the wheels and the paper, motor error, pen-rule human vision accuracy, etc. To demonstrate that we did 5 lines of 10 cm going at 2 cm/s and all the lines are different. So there is some kind of error/noise.



When measuring the angle we have seen that there is no pattern and it does not change or saturates when velocity is above 3.8 cm/s. This is what we were expecting as the angular velocity does not depend on the linear velocity. We also have some kind of error/noise on the angles because of the same reasons as said above when measuring the length.

## **5. Summary of teammate contributions**

We worked together for all the homework tasks. The contribution in this homework has been equal for the both of us.