EE 585 Probabilistic Robotics: Assignment 1

Umut Kurt

November 11, 2022

1 Review and Use of a Robot Simulator

1.1 Ability to simulate the kinematics and/or dynamics of a differential drive (wheeled) robot

Before starting to simulate a differential drive robot, we should be familiar with some of the key concepts and terminology of the Webots. In Webots simulation environment, the different objects are called *Nodes* and organized hierarchically in a *Scene Tree*. A node may contain different sub-nodes, that can be set up by the user, corresponding to the characteristic of the node. For example *Solid Nodes* are used to represent objects with physical properties such as dimension, mass and contact material. The solid class is the base class for collision-detected objects. Robot class is a subclass of the solid class.

Webots offers a variety of off-the-shelf robot models that can be imported from *PROTO nodes*. These robot models comes with built-in actuators, sensors and default controllers. Alternatively, a robot node can be used to built a differential drive robot model from scratch. Fundamental structure of the robot consists of two *HingeJoint*, *RotationalMotor* and *PositionalSensor* nodes to simulate dynamics of a differential drive robot. In order to create a model of a differential robot, first a *Robot* node is added. Then a *Shape* node with box geometry is added to robot node, which is the body of the differential robot. The wheels of the robot are attached to the body by *HingeJoint* nodes with cylindrical *Solid* endpoints. *HingeJoint* node offers three devices, in this case *RotationalMotor* and *PositionalSensor* will be used. For robot to stay in balance a caster wheel will be modeled. A caster wheel can be created by adding a ball joint with a *Shape* node as a children. In the *Shape* node, geometry is selected as sphere. Finally, *Physics* node is very important for simulating a differential robot. If a robot contains a *Physics* node, simulation will take into account the forces acting on solid bodies and the physical properties of these bodies. If the *Physics* node is NULL, the motion will be simulated using kinematics mode which negates the forces that cause the motion and body will be treated as massless. The 3D model of the basic differential robot is shown in Figure 1.

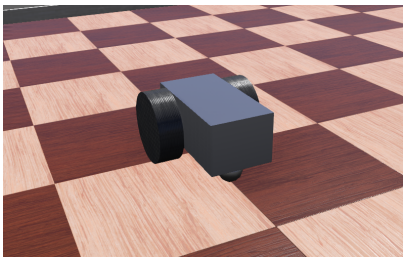


Figure 1: 3D model of the differential drive robot.

1.2 Ability to command the linear and angular velocities of the robot

In order to command the linear and angular velocities of the robot, a controller is required. A controller is a program that is executed by the simulation engine at each time step. The actions of the robot is determined by the controller. In this assignment, controllers are written in Python language. Different controllers can be selected from the controller section inside the *Robot* node. A linear velocity command can be given by setting the same velocities to both wheels. Linear velocity can be calculated by multiplying the velocity and wheel radius. In the controller, first an instance of the robot class is initialized. Then the *RotationalMotor* devices are called using *getDevice* method

and enabled. The motors can be controlled eiter by setting the target velocity or by setting the target position using *setVelocity* and *setPosition* methods respectively. The example controller code is shown below. Further information about the controller can be found in the Webots documentation.

```
from controller import Robot, Motor
# create the Robot instance.
robot = Robot()
# Initialize motors
motors = []
motorNames = ['left_wheel', 'right_wheel']
for i in range(len(motorNames)):
    motors.append(robot.getDevice(motorNames[i]))
    motors[i].setPosition(float('inf'))
    motors[i].setVelocity(0.0)
# get the time step of the current world.
timestep = int(robot.getBasicTimeStep())
# Main loop:
# - perform simulation steps until Webots is stopping the controller
while robot.step(timestep) != -1:
    #set wheel velocity to 1 m/s
    for i in range(len(motorNames)):
        motors[i].setVelocity(1)
    pass
```

During simulation, the linear velocity of the robot can be observed selecting robot. When the robot is selected four tabs appear below the *Scene Tree* . Both linear and angular velocities of the robot in 3-axes can be observed in the *Velocity* tab. Since wheel radius is 0.12 m and the commanded velocity is 1 m/s expected linear velocity is 0.12 m/s. The linear velocity of the robot is shown in Figure 2.

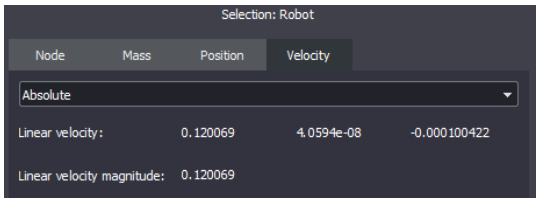


Figure 2: Linear velocity of the robot.

The angular velocity command can be given by setting different velocities to the wheels. As an example the left wheel is set to 1 m/s and the right wheel is set to -1 m/s. Robot turns around on the axis at the midpoint of the two differential wheels. The angular velocity of the robot can be calculated by dividing linear velocities of the wheels by the turning radius which is the half of the distance between wheels, in this case 0.18 m. This corresponds to $\frac{0.12}{0.18} = 0.66$ m/s The angular velocity of the robot is shown in Figure 3.

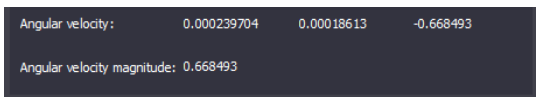


Figure 3: Angular velocity of the robot.

1.3 Ability to simulate ”real” noise into the motion kinematics/dynamics of the robot

In order to simulate ”real” noise into the motion kinematics/dynamics of the robot, two different aprroaches applied together. First approach is to add random normal noise to velocity commands in the controller level. Second approach is to add *ContactProperties* node for the wheels and modelling wheel slip. In order to model slip, the material of the wheels is stated and then in the *WorldInfo* tab a *ContactProperties* node is added for wheel material. The *ContactProperties* node is used to define the friction coefficient between two materials. An assymetric coulomb friction is used to model slip. The friction coefficient is set to 0.4. To illustrate the effect of slip, and the noise added to velocity command, the robot is commanded to move in a straight line for 0.3 m. The simulation is run for 100 times. The resulting positions of the robot is recorded using a GPS device and the output data is plotted in MATLAB. The results are shown in Figure 4.

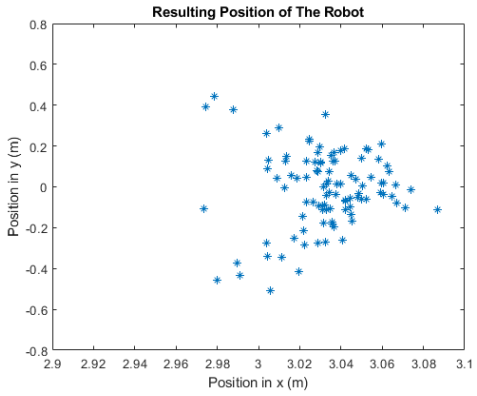
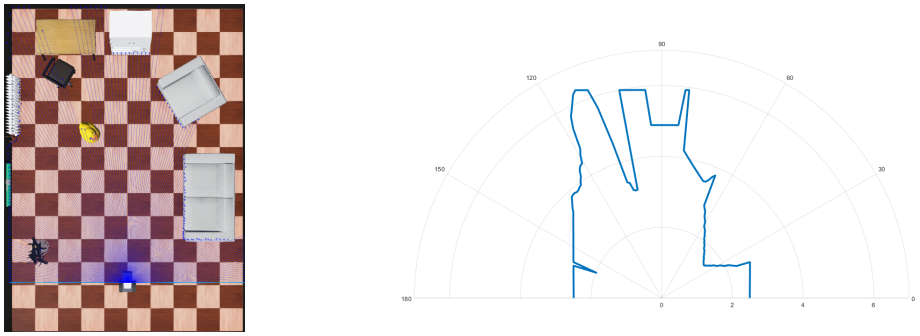


Figure 4: Motion noise.

1.4 Distance Sensor Data

As the distance sensor, a lidar ’Sick LMS 291’ is used. Alternatively, a lidar node can be created from scratch. This device is added to the *Robot* node, from *PROTO nodes(Webots Projects)*, the sensor resolution is set to 180 which corresponds to a precision of 1° . In order to get the distance sensor data, sensor should be called using *getDevice*, then enabled and finally the distance data can be obtained using *getRangeImage* method. Data returns a list of distances, indexed by the angle of the corresponding sensor ray. The data is later saved to a csv file and plotted in MATLAB. The simulation environment and polar plot corresponding to the distance sensor data is shown in Figure 5. Further information about the lidar sensor can be found in the *Webots Reference Manual* documentation.



(a) The simulation environment. (b) Polar plot of the distance sensor data.

Figure 5: Lidar sensor data in Webots Simulation.

1.5 Defining map environment

Webots, offers a variety of objects below *Proto Nodes*. These objects can be used to create a map environment. In this assignment, a map environment is created by adding typical living room objects to the *Scene Tree*. The map environment is shown in Figure 6. Alternatively different obstacles or landmarks can be placed in the simulation environment using *Solid* nodes with *boundingObjects*.

Figure 6: Map environment.

1.6 Odometry Data

As mentioned before, differential drive robot includes two *PositionalSensor*, these devices simulates encoder. Odometry data can be get from these sensors. In order to get the odometry data, sensors should be called using *getDevice*, then enabled and finally the distance data can be obtained using *getValue* method. Note that the odometry data is in radians, if the data is needed in meters, it should be converted by multiplying the data with the wheel radius. However, this calculation is only valid for linear motion. To compute the position of the robot from encoder data we need to take into account both the angular velocity and linear velocity of the robot using equations below and integrate them.

$$v = \frac{v_{left} + v_{right}}{2} \tag{1}$$

$$w = \frac{v_{left} - v_{right}}{d} \tag{2}$$

$$\dot{\theta} = w \tag{3}$$

$$\dot{x} = v\cos(\theta) \tag{4}$$

$$\dot{y} = v\sin(\theta) \tag{5}$$

$$\tag{6}$$

Where d is the distance between two wheels, v_{left} and v_{right} are the linear velocities of the respective wheels.
In order to visualize odometry data, robot is moved in a square path with 1.5 m side length. The odometry data is saved to a txt file and plotted in MATLAB. For comparison, the robot is equipped with a GPS sensor. The GPS sensor data is also saved to a txt file and added to same plot. The GPS and the odometry data is shown in Figure 7. The odometry data is shown with blue '$*$', GPS data is shown in orange circles. Even without noise in the odometry data there are small discrapancies between the two data sets. This is due to numerical errors introduced from the odometry data calculations. A more detailed information on position sensors can be found in the *Webots Reference Manual* documentation.

1.7 Camera Data

A *Camera* node is added as a children node to the robot. Various aspects of the camera can be modified from the camera setups. Such as resolution, field of view, noise amd spherical distortion. When camera is enabled from

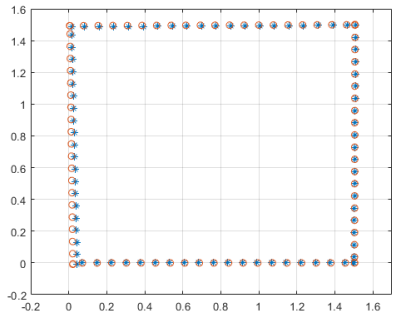


Figure 7: Odometry and GPS data.

the controller, the resulting image is displayed in 3D window. The camera pixel informations can be obtained using *getImage* method. The image array is converted to RGB and saved to as a png file. The camera image is shown in Figure 8. A more detailed information on camera nodes can be found in the *Webots Reference Manual* documentation.

Figure 8: Camera image.

1.8 Simulating sensor noise

The lidar, position sensor and camera nodes offer noise parameters. These parameters can be used to simulate noise in the sensor data. All the sensors are modified to have noise parameters. In the Figure 9, the noisy lidar data is given in comparison with the previous noiseless data in Figure 5b. The noise is set to 0.2. The noisy data is shown in orange and noiseless data is shown in blue.

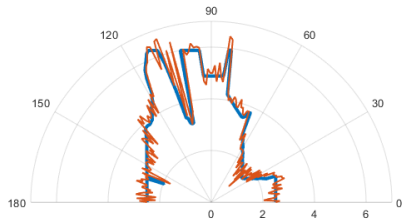


Figure 9: Noisy lidar data compared to noiseless data.

The odometry data from noisy position sensor is shown in Figure 10. The noise is set to 0.2 for the position sensor

on the each wheel. The data is collected in a square path with 1.5 m side length and compared with noiseless GPS data as the ground truth. The noisy odometry data is shown in blue and noiseless GPS data is shown in orange. There is a significant difference between the odometry and GPS data due to noise.

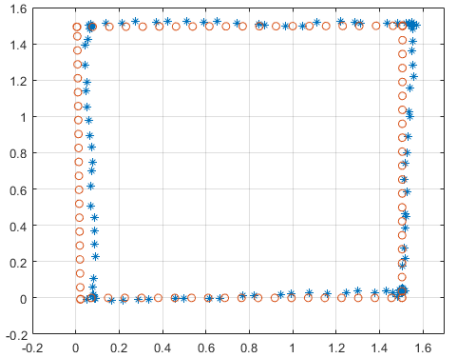


Figure 10: Noisy odometry data compared to ground truth GPS data.

The camera noise is set to 0.2 and the resulting image is shown in Figure 11. The effect of noise on the image pixels is visible in the image when compared to the noiseless image in Figure 8.

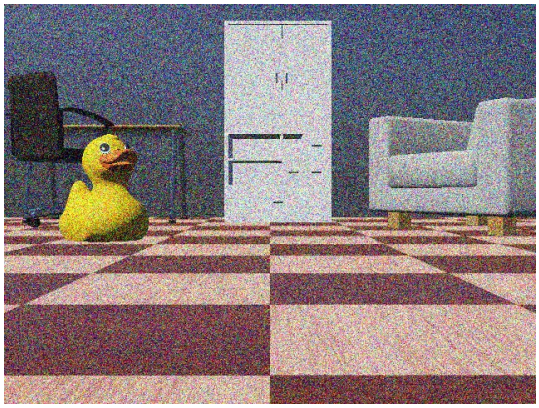


Figure 11: Noisy camera image.

2 Robot Following Task

In the following task, camera sensor is used for identifying and tracking the leading robot. The leading robot is fitted with a green spherical solid at the back bumper. The controller of the robot behind the leading robot takes camera image and extracts the contour of the green ball by using OpenCV library. The center of the green ball is calculated and the controller uses a proportional controller to steer the differential drive to keep the ball in the center of the image. The tracking distance is set to 0.65 m. To keep the distance stable, the controller uses another proportional controller to set the linear velocity of the robot. The radius of the ball is used as a measure of distance to the leading robot. The distance between two robots is shown in Figure 12.
The controller performs following task with a bounded tracking distance fluctuating between 0.6 m and 0.7 m. A sample video of the robot following task is available online.

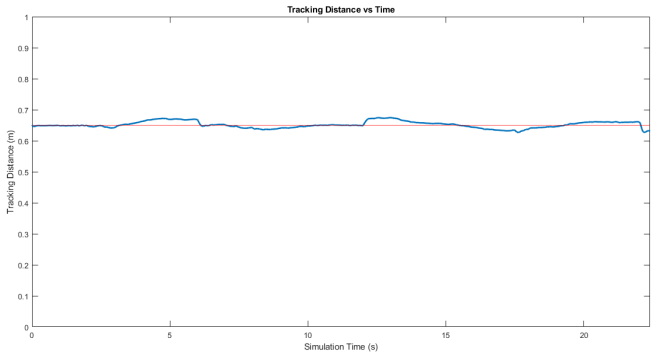


Figure 12: Tracking Distance vs Time.