Sim.I.am: A Robot Simulator

Coursera: Control of Mobile Robots

Jean-Pierre de la Croix

Last Updated: January 28, 2013

Manual

This manual is going to be your resource for using the simulator in the programming exercises for this course. It will be updated throughout the course, so make sure to check it on a regular basis. You can access it anytime from the course page on Coursera under `Programming Exercises`.

Installation

Download `simiam-coursera-week-X.zip` (where X is the corresponding week for the exercise) from the course page on Coursera under `Programming Exercises`. Make sure to download a new copy of the simulator **before** you start a new week's programming exercises, or whenever an announcement is made that a new version is available. It is important to stay up-to-date, since new versions may contain important bug fixes or features required for the programming exercises.

Unzip the `.zip` file to any directory.

Requirements

You will need a reasonably modern computer to run the robot simulator. While the simulator will run on hardware older than a Pentium 4, it will probably be a very slow experience. You will also need a copy of MATLAB. The simulator has been tested with MATLAB R2011a, so it is recommended that you use that version or higher.

Bug Reporting

If you run into a bug (issue) with the simulator, please leave a message in the discussion forums with a detailed description. The bug will get fixed and a new version of the simulator will be uploaded to the course page on Coursera.

Mobile Robot

The mobile robot platform you will be using in the programming exercises is the Khepera III (K3) mobile robot. The K3 is equipped with 11 infrared (IR) range sensors, of which nine are located in a ring around it and two are located on the underside of the robot. The IR sensors are complemented by a set of five ultrasonic sensors. The K3 has a two-wheel differential drive with a wheel encoder for each wheel. It is powered by a single battery on the underside and can be controlled via software on its embedded Linux computer.

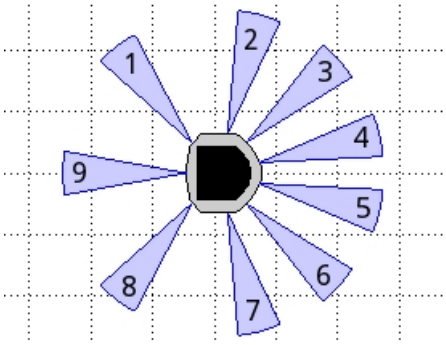

Figure 1: IR range sensor configuration

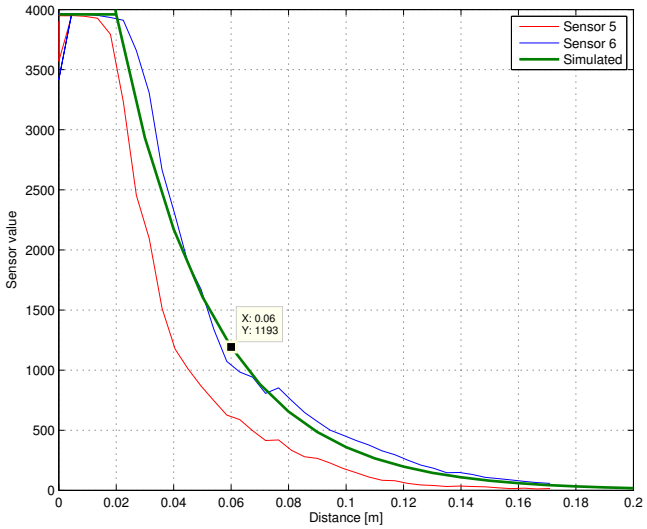

Figure 2: Sensor values vs. Measured Distance

IR Range Sensors

For the purpose of the programming exercises in the course, you will have access to the array of nine IR sensors that encompass the K3. IR range sensors are effective in the range 0.02 m to 0.2 m only. However, the IR sensors return raw values in the range of $[18, 3960]$ instead of the measured distances. Figure 2 demonstrates the function that maps these sensors values to distances.

The green plot represents the sensor model used in the simulator, while the blue and red plots show the sensor response of two different IR sensors (under different ambient lighting levels). The effect of ambient lighting (and other sources of noise) are **not** modelled in the simulator, but will be apparent on the actual hardware.

The function that maps distances, denoted by Δ, to sensor values is the following piecewise function:

$$f(\Delta) = \begin{cases} 3960, & \text{if } 0\text{m} \leq \Delta \leq 0.02\text{m} \\ \lfloor 3960e^{-30(\Delta - 0.02)} \rfloor, & \text{if } 0.02\text{m} \leq \Delta \leq 0.2\text{m} \end{cases} \tag{1}$$

Your controller can access the IR array through the `robot` object that is passed into the `execute` function. For example,

```
for i=1:9
   fprintf('IR #%d has a value of %d.\n', i, robot.ir_array(i).get_range());
end
```

The orientation (relative to the body of the K3, as shown in figure 1) of IR sensors 1 through 9 is $128°, 75°, 42°$, $13°, -13°, -42°, -75°, -128°$, and $180°$, respectively.

Ultrasonic Range Sensors

The ultrasonice range sensors have a sensing range of 0.2m to 4m, but are not available in the simulator.

Differential Wheel Drive

Since the K3 has a differential wheel drive (i.e., is not a unicyle), it has to be controlled by specifying the rotational velocities of the right and left wheel. These velocities are computed by a transformation from (v, ω) to (v_r, v_ℓ). Recall that the dynamics of the unicycle are defined as,

$$\begin{aligned} \dot{x} &= vcos(\theta) \\ \dot{y} &= vsin(\theta) \\ \dot{\theta} &= \omega. \end{aligned} \tag{2}$$

The dynamics of the differential drive are defined as,

$$\begin{aligned} \dot{x} &= \frac{R}{2}(v_r + v_\ell)cos(\theta) \\ \dot{y} &= \frac{R}{2}(v_r + v_\ell)sin(\theta) \\ \dot{\theta} &= \frac{R}{L}(v_r - v_\ell), \end{aligned} \tag{3}$$

where R is the radius of the wheels and L is the distance between the wheels.

The speed of the K3 can be set in the following way assuming that you have implemented the `uni_to_diff` function, which transforms (v, ω) to (v_r, v_ℓ):

```
v = 0.15; % m/s
w = pi/4; % rad/s
% Transform from v,w to v_r,v_l and set the speed of the robot
[vel_r, vel_l] = obj.robot.dynamics.uni_to_diff(robot,v,w);
```

Wheel Encoders

Each of the wheels is outfitted with a wheel encoder that increments or decrements a tick counter depending on whether the wheel is moving forward or backwards, respectively. Wheel encoders may be used to infer the relative pose of the robot. This inference is called **odometry**. The relevant information needed for odometry is the radius of the wheel, the distance between the wheels, and the number of ticks per revolution of the wheel.

```
R = robot.wheel_radius; % radius of the wheel
L = robot.wheel_base_length; % distance between the wheels
tpr = robot.encoders(1).ticks_per_rev; % ticks per revolution for the right wheel

fprintf('The right wheel has a tick count of %d\n', robot.encoders(1).state);
fprintf('The left wheel has a tick count of %d\n', robot.encoders(2).state);
```

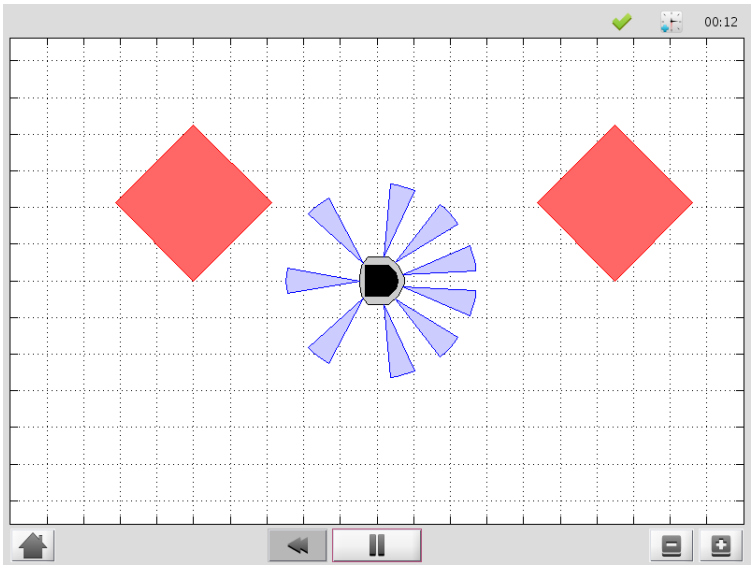

Figure 3: Simulator

Simulator

Start the simulator with the `launch` command in MATLAB.

Figure 3 is a screenshot of the graphical user interface (GUI) of the simulator. The GUI can be controlled by the bottom row of buttons (or their equivalent keyboard shortcuts). The first button is the *Home* button `[h]` and returns you to the home screen. The second button is the *Rewind* button and resets the simulation. The third button is the *Play* button `[p]`, which can be used to play and pause the simulation. The set of *Zoom* buttons `[[,]]` or the mouse scroll wheel allows you to zoom in and out to get a better view of the simulation. The set of *Pan* buttons `[left,right,up,down]` can be used to pan around the environment, or alternatively, Clicking, holding, and moving the mouse allows you to pan too. *Track* button `[c]` can be used to toggle between a fixed camera view and a view that tracks the movement of the robot. You can also simply click on a robot to follow it.

Programming Exercises

The following sections serve as a tutorial for getting through the simulator portions of the programming exercises.

Week 2

The first week's programming exercises ask you to implement odometry in the `+simiam/+controller/+khepera3/K3Supervisor.m` file, as well as, the transformation from (v, ω) to (v_r, v_ℓ) in the `+simiam/+robot/+dynamics/DifferentialDrive.m` file. Since odometry and this particular transformation will be used in all of your controllers, make sure to implement these correctly.

To test odometry and your transformation, you will likely want to edit `+simiam/+controller/GoToGoal.m` and implement a P-regulator. Similarily, for converting raw IR values to distances, edit `+simiam/+controller/AvoidObstacle.m`.

Note: The code uses (x, y, θ) to represent the pose in 2D, while the video lectures refer to (x, y, ϕ). Both θ (`theta`) and ϕ are simply symbols that represent the orientation of the robot.