MATLAB Communication Toolbox for the iRobot Roomba

User manual

# Disclaimer

This toolbox is based on the original toolbox made for serial communication, by Joel Esposito, Associate Professor (US Naval Academy). The communication settings were changed so that the commands and sensor data are send over UDP, to a single board computer mounted on the iRobot Roomba (Raspberry Pi). The unit conversions (S.I units to bytes for the Roomba and vice versa) are unchanged.

The syntax of the functions are changed, and new functions were made to adapt the student needs.

Important note: the build in functions for odometry of the Roomba (distance and angle) were unreliable. The toolbox functions of odometry are now based on the raw encoder values.

Disclaimer of the original simulator toolbox:

--------------------------------------------------------------------------------------------------------------------------------------

Copyright (c) 2011, Joel Esposito

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

\* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution

Esposito, Joel M., Barton, Owen, Koehler, Joshua, Lim, David, "Matlab Toolbox for the Create Robot", www.usna.edu/Users/weapsys/esposito/roomba.matlab/, Copyright 2008-2011

--------------------------------------------------------------------------------------------------------------------------------------

# A word about the iRobot Roomba

The iRobot Roomba is designed to be a low-cost, off-the-shelf automatic vacuum cleaner. But why is it used more and more in an educational or research context, where its vacuuming functions aren’t even used? Firstly, the iRobot Roomba is one of the cheapest robotic off-the-shelf products. Secondly, and most importantly, iRobot shared the command interface to control the Roomba with a serial cable.

# Purpose of the Communication Toolbox

The toolbox replaces the native low-level numerical commands, with a set of high level, intuitive, MATLAB functions that:

• Create a link between your PC and the Pi using UDP protocol. The Pi is connected to the Roomba with a serial cable, and will transfer all data between the PC and the Roomba.

• Provide a variety of user-friendly drive commands, calibrated in SI units.

• Read the bump and cliff sensors, determine distance driven, and battery life in SI units.

• Use the MATLAB command line or script files to control the robot. Note that all code is developed, stored and executed on the PC base station, not the Raspberry Pi.

• Receive the Lidar data send from the Raspberry Pi

# Installation

1. Download on Toledo the .zip
2. Unzip the toolbox to extract the toolbox folder containing all the functions.
3. Place that folder in a permanent location on your computer’s hard disk drive.
4. Start MATLAB
5. In the menu toolbar, click File, then select Set Path.
6. Click Add Folder with Subfolders, navigate to the toolbox folder you just saved, select it, then press OK.
7. Click the Save button, then press Close.

# Writing the Autonomous Control Program

## Requirements

The control program must with initRoomba(IP). The output, comPort, is used all the function needed to control the Roomba, and to ask for sensor data.

MATLAB code 1:

startup\_RoombaUDP

roombaName='Roomba0';

IP=getIPRoomba(roombaName);

comPort=initRoomba(IP);

[BumpRight BumpLeft WheDropRight WheDropLeft WheDropCaster ...

BumpFront] = BumpsWheelDropsSensorsRoomba(comPort)

## Recommendations

Most control programs will have the majority of their functions contained in a loop. Inside will be a number of conditionals that will allow the robot to react to particular events. For example, the program will call getSafetySensors. If a bump sensor is activated, the robot will back up and turn, before moving forward again. This very simplified; the real control program could specify more complex behavior based on multiple inputs. For one way to structure such a program, see the file ExampleControlProgram.m.

# Differences between the Communication and Simulation toolbox

The workaround in the simulator, to keep the same syntax between the communication toolbox and the simulation toolbox is to write your control program, in a function, which reserves as input, from the simulator, the comPort. Now, this has to be done at the start of your control program, with initRoomba.

The LIDAR communication is slightly different. In your simulation, it expects the same comPort, as the Roomba, but here, it reserves a different port, called localPort, which is the port were the Raspberry Pi send the LIDAR data to.

# Globals

This MATLAB toolbox makes use of globals (variables accessible at any time, in any script/function). This means that there are a few variable names that you can’t use, otherwise this will interfere with the already assignment global values. The variable names that you can’t use are:

* comDelay
* sensorSkip
* LeftEncoderCount
* RightEncoderCount

# Tips and tricks

This section will give you some general tips and tricks, to avoid problems when using this toolbox. If something goes wrong try to find if this problem is already described here.

## Concerning the Roomba

### Slow communication with Roomba

Sometimes, there is a slow communication between the Roomba and your PC, this resulting in failures to receive correct measurements. If this happens, try to reboot your PC and the Pi and launch: launchTest\_Roomba. The resulting frequency of this test script should be +/- 11 Hz.

### No beep at start

Not hearing beep at start = not a good initialization of the Roomba communication. Crtl+c to end you program and execute it again!

### Odometry information

Don’t forget, odometry data is shared over 3 functions. All these functions will update the globals Encoder. This resulting, in the distance/angle since last call of ONE of those 3 functions. Example

MATLAB code 2:

|  |  |
| --- | --- |
| startup\_RoombaUDP  roombaName='Roomba0';  IP=getIPRoomba(roombaName);  comPort=initRoomba(IP);  setFwdVelRadius(comPort,0.5,inf)  pause(1)  setFwdVelRadius(comPort,0,inf)  [ds,~]=getOdometry(comPort) ----------------------------------  OUTPUT :  ds = 0.5 | startup\_RoombaUDP  roombaName='Roomba0';  IP=getIPRoomba(roombaName);  comPort=initRoomba(IP);  travelDist(comPort,0.5,0.5)  [ds,~]=getOdometry(comPort) ----------------------------------  OUTPUT :  ds = 0 |

## Concerning the LIDAR

### Initializing takes time!

The initialization between the Pi and the LIDAR is quiet slow. A first indicator when the initialization is completed is when the LIDAR stops turning. Still, wait a few seconds (5-10) before executing a script calling the LIDAR (initLidar). This could lead towards a miscommunication, and you will have to reboot the Pi.

Once you have used the initLidar function, the Lidar will start and continuously sending information towards the PC. This will not stop until you turn off the Pi.

# Function specification

## Initialization functions

### getIPRoomba

IP = getIPRoomba(roombaName)

#### Summary

This function gives you the IP to communicate with a certain Roomba. In other words, the static IP of the Pi, which is connected to the Roomba. Each Roomba has a name, which has to be given as a string.

#### Input

|  |  |
| --- | --- |
| roombaName | String, name of the Roomba you want to control |

#### Output

|  |  |
| --- | --- |
| IP | Static IP of the Pi connected to the selected Roomba |
|  |  |

### initRoomba

comPort = initRoomba(IP)

#### Summary

This function initializes the communication between the PC, Pi and Roomba. It will set the Roomba in the right mode and make a beep sound to indicate the user that the initialization of the Roomba is successful. This function will return an object, comPort, which has to be used in every following function. The comPort contains all the information needed to communicate with the Raspberry Pi.

#### Input

|  |  |
| --- | --- |
| IP | Static IP of the Roomba you want to control. User user-friendly getIPRoomba function |

#### Output

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |
|  |  |

### initLidar

localPort=initLidar(IP)

#### Summary

This function initializes the communication between the PC, Pi and Lidar.

#### Input

|  |  |
| --- | --- |
| IP | Object containing all the necessary information to communicate with the Raspberry Pi |

#### Output

|  |  |
| --- | --- |
| localPort | Local port on which the Pi will send all the lidar data information |

## Sensor functions

### getCliffSensors

[leftCliff, leftFrontCliff, rightFrontCliff, rightCliff] = getCliffSensors(comPort)

#### Summary

The function will display the state of all the cliff sensors, either triggered or not triggered.

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |

#### Output

|  |  |
| --- | --- |
| dist | Double, meters travelled since last call to getOdometry/travelDist/turnAngle. The value saturates at ± 32 m |
| angle | Double, radians turned since last call to getOdometry/travelDist/turnAngle. The value saturates at ± 571 rad. |

### getInternalSensors

[chargeState,voltage,current,temp,pCharge] = getInternalSensors(comPort)

#### Summary

The function reads all the available internal sensors of the Roomba.

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |

#### Output

|  |  |
| --- | --- |
| chargeState | Int, |
| voltage | Double, voltage of the battery [0-65535] mV |
| current | Double, [-32768 – 32767] mA |
| temp | Double, temperature of the Roomba, [-128 – 127] °C |
| pCharge | Double, battery percentage, 100% means fully charged, [0-100] % |

### getOdometry

[dist,angle] = getOdometry(comPort,v,r)

#### Summary

The function uses the odometry to approximate the relative displacement and rotation since last call of this function, or travelDist and turnAngle. Call this function regularly to avoid saturation (max distance = 32m and max rotation = 571 rad).

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |
| v | Current velocity of the Roomba [m/s] |
| r | Current radius of the Roomba [m] |

#### Output

|  |  |
| --- | --- |
| dist | Double, meters travelled since last call to getOdometry/travelDist/turnAngle. The value saturates at ± 32 m |
| angle | Double, radians turned since last call to getOdometry/travelDist/turnAngle. The value saturates at ± 571 rad. |

### getSafetySensors

[BumpRight BumpLeft WheDropRight WheDropLeft WheDropCaster ...  
BumpFront] = BumpsWheelDropsSensorsRoomba(comPort)

#### Summary

This function will read the bump and wheel drop sensors on the robot. On the real Create, the wheel drop sensors will activate if the wheels move over empty space and fall to their extended position. In the simulator, there are no cliffs so the wheel drop sensors will always remain inactive. The bump sensors work similarly for both the simulator and the real Create.

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |

#### Output

|  |  |
| --- | --- |
| BumpRight | Double of Boolean value, 1 if the right bump sensor is pressed, 0 if not. |
| BumpLeft | Double of Boolean value, 1 if the left bump sensor is pressed, 0 if not. |
| WheDropRight | Double of Boolean value, 1 if the right wheel is in the extended position, 0 if not. The simulator will always return 0. |
| WheDropLeft | Double of Boolean value, 1 if the left wheel is in the extended position, 0 if not. The simulator will always return 0. |
| WheDropCaster | Double of Boolean value, 1 if the caster wheel is in the extended position, 0 if not. The simulator will always return 0. |
| BumpFront | Double of Boolean value, 1 if the front bump sensor is pressed, 0 if not. |

### getWallSensor

wall = getWallSensor(comPort)

#### Summary

This will return state of the IR sensor, at the front right side of the Roomba. This is a Boolean value, 1 if there is a wall at +/- 10cm, 0 if there is no wall

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |

#### Output

|  |  |
| --- | --- |
| wall | Boolean, indicator if there is a wall at the front right side of the robot. |

### getLidarSensor

[dist,angle] = getLidarSensor(localPort)

#### Summary

This function will return distance readings for each of the points of measurement on the LIDAR sensor. This will be a very long array, and will take a few seconds to compute. It would be best if the robot were not moving during this function call.

#### Input

|  |  |
| --- | --- |
| localPort | Port on the PC were the LIDAR is sending its information. |

#### Output

|  |  |
| --- | --- |
| dist |  |
| angle |  |

## Actuator functions

### setDriveWheels

setDriveWheels(comPort,rightVel,leftVel)

#### Summary

This command will set the speed of the individual drive wheels (left and right) of the Roomba.

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |
| rightVel | Velocity of the right wheel of the Roomba in meters per second. Positive values indicate forward movement, negative indicates backwards. Values must be between -0.5 and 0.5 m/s. |
| leftVel | Velocity of the left wheel of the Roomba in meters per second. Positive values indicate forward movement, negative indicates backwards. Values must be between -0.5 and 0.5 m/s. |

### setFwdVelAngVel

setFwdVelAngVel(comPort,FwdVel,AngVel)

#### Summary

This command will set the overall forward and angular velocities of the Create.

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |
| FwdVel | Linear velocity of the Create in meters per second. Positive values indicate forward movement, negative indicates backwards. Values must be between -0.5 and 0.5 m/s. |
| AngVel | Angular velocity of the Create in radians per second. Positive values indicate counter-clockwise rotation, negative indicates clockwise. Values must be between -2.5 and 2.5 rad/s. |

### setFwdVelRadius

setFwdVelRadius(comPort,FwdVel,Radius)

#### Summary

This command will move the robot in an arc with the specified radius. This is useful for circular and spiral-type movement paths, or making smooth turns. See below for special-case inputs.

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |
| FwdVel | Double, linear velocity of the robot in meters per second. Positive values indicate forward movement, negative values indicate reverse. Values must be between -0.5 and 0.5 m/s. |
| Radius | Double, turning radius for the robot path in meters. Positive values indicate counter-clockwise turning, negative values go clockwise. Special cases are listed below. Values must be between -2 and -eps m or eps and 2 m. eps is a MATLAB built-in variable indicating the lowest value above 0. This is done to ensure that a turning direction is known.   |  |  | | --- | --- | | inf | Travel in a straight line | | eps | zero-point turn (turn in place) counter-clockwise | | -eps | zero-point turn clockwise | |

### travelDist

travelDist(comPort,speed,distance)

#### Summary

This command will make the robot move the specified distance in a straight line at the specified speed. The function will continue to execute until the movement is complete, so no other functions will be available during that time. The distance traveled is dependent on the odometry, so it will not be exact if the odometry sensor is noisy. This function is not recommended for frequent use.

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |
| speed | Double, linear speed at which the robot moves in meters per second. This value must be positive, between 0.025 and 0.5 m/s. |
| distance | Double, distance the robot will travel before stopping in meters. This also controls the direction of movement, positive moves forward, negative moves backwards. |

### turnAngle

turnAngle(comPort,speed,angle)

#### Summary

This command will make the robot turn the specified angle at the specified speed. The function will continue to execute until the movement is complete, so no other functions will be available during that time. The angle turned is dependent on the odometry, so it will not be exact if the odometry sensor is noisy. This function is not recommended for frequent use.

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |
| speed | Double, speed at which the robot turns in radians per second. This value must be positive, between 0 and 0.2 rad/s. |
| angle | Double, angle to turn in degrees. The value must be between -360° and 360°. In actuality, this is more like the angle to end up at. The robot will turn to take the shortest path to the specified angle. So commands between 0° and 180° and between -180° and -360° will turn counter-clockwise. Commands between 180° and 360° and between 0° and -180° will turn clockwise. |

### stopRoomba

stopRoomba(comPort)

#### Summary

Stops the Roomba, and puts it in default modus (start)

#### Input

|  |  |
| --- | --- |
| comPort | Object containing all the necessary information to communicate with the Raspberry Pi |

# Examples

## Example 1

controlProgram1 will let the Roomba drive for 5 seconds, at 0.5 m/s, without taking into account collisions

MATLAB code 3:

IP=getIPRoomba(‘Roomba0’)

comPort=initRoomba(IP);

% Parameters

FwdVel = 0.5; % m/s

AngVel = 0; % rad/s

setFwdVelAngVel(comPort,FwdVel,AngVel);

pause(5)

FwdVel = 0; % speed, m/s (stop)

setFwdVelAngVel(comPort,FwdVel,AngVel);

## Example 2

controlProgram2 will let the Roomba drive for 5 seconds, at 0.5 m/s, taking into account collisions. If the Roomba collides with a wall or object, it will stop.

MATLAB code 4:

IP=getIPRoomba(‘Roomba0’)

comPort=initRoomba(IP);

% Parameters

FwdVel = 0.5; % m/s

Radius = inf; % rad/s

setFwdVelRadius(comPort,FwdVel,Radius);

[BumpRight, BumpLeft,BumpFront, ~, ~] = getSafetySensors(comPort);   
% reading bump sensors

bumps = BumpRight + BumpLeft + BumpFront;   
% combining all bump sensors. + is OR

tic % start timer. toc is just reading the time since tic.

while toc < 5 && bumps ~= true % && is and

[BumpRight, BumpLeft,BumpFront, ~, ~] = getSafetySensors(comPort);

bumps = BumpRight + BumpLeft + BumpFront;

end

FwdVel = 0; % speed, m/s (stop)

setFwdVelRadius(comPort,FwdVel,Radius)