MATLAB Simulator for the iRobot Create

Function Specifications

Author: Cameron Salzberger
Advisors: Dr. Hadas Kress-Gazit
Dr. K-Y Daisy Fan
Supported By: The MathWorks
Updated: 2/18/2013 by Kevin Wyffels

CORNELL UNIVERSITY

Contents

Disclaimer

Overview

This document assumes you have read the User Manual for the simulator toolbox. This is intended to provide descriptions of all simulator functions that the autonomous control program may call. This document is duplicates much of the "`help`" comments at the beginning of each function. The majority of these functions have the same input/output specifications as those in the MATLAB Toolbox for the iRobot Create (MTIC), but there a few new ones for the User-Added sensors (sonars, LIDAR, camera, and overhead localization). The functions should work equivalently on the real Create, assuming the hardware is installed the same. See the MTIC Documentation to double-check those functions:
http://www.usna.edu/Users/weapsys/esposito/roomba.matlab/Matlab_Toolbox_iRobot_create_doc.pdf

The sonar sensor functions are designed to work with the WOOSH board that interfaces between the Bluetooth Adaptor Module (BAM) and the cargo bay connector on the Create. The sonar functions are specific to that board's firmware. The LIDAR, camera, and overhead localization functions are specific to a unique setup at Cornell, and will probably require adaptation to use anywhere else. The WOOSH board specifications can be found at the site below:
http://tomyumcorp.com/whoosh

AllSensorsReadRoomba

```
[BumpRight BumpLeft BumpFront Wall virtWall CliffLft ...
 CliffRgt CliffFrntLft CliffFrntRgt LeftCurrOver ...
 RightCurrOver DirtL DirtR ButtonPlay ButtonAdv Dist ...
 Angle Volts Current Temp Charge Capacity pCharge]= ...
AllSensorsReadRoomba(obj)
```

Summary

On the real Create this function reads a series of sensors in a row. Because of the data streaming serial communication capabilities on the real Create, this function will execute faster than reading each of the sensors individually. On the simulator there is no similar advantage. If the control program is designed for the simulator only, it is recommended to only read the sensors it requires.

The sensors read by this function are explained more fully in the User Guide on pages 7-10.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

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.
`BumpFront` – Double of Boolean value, 1 if the front bump sensor is pressed, 0 if not.
`Wall` – Double of Boolean value, 1 if a wall is detected next to the wall sensor, 0 if not.
`virtWall` – Double of Boolean value, 1 if a virtual wall is detected by the IR receiver, 0 if not.
`CliffLft` – Double of Boolean value, 1 if a cliff is detected under the left cliff sensor, 0 if not. The simulator will always return 0.
`CliffRgt` – Double of Boolean value, 1 if a cliff is detected under the right cliff sensor, 0 if not. The simulator will always return 0.
`CliffFrntLft` – Double of Boolean value, 1 if a cliff is detected under the front left cliff sensor, 0 if not. The simulator will always return 0.
`CliffFrntRgt` – Double of Boolean value, 1 if a cliff is detected under the front right cliff sensor, 0 if not. The simulator will always return 0.
`LeftCurrOver` – Double of Boolean value, 1 if the left motor is receiving more than 1 A of current, 0 if not. The simulator will always return 0.
`RightCurrOver` – Double of Boolean value, 1 if the right motor is receiving more than 1 A of current, 0 if not. The simulator will always return 0.
`DirtL` – Double, always 0 since there are no dirt sensors in either the real Create or the simulator.
`DirtR` – Double, always 0 since there are no dirt sensors in either the real Create or the simulator.

`ButtonPlay` – Double of Boolean value, 1 if the Play (**>**) button is pushed, 0 if not. Note the difference between button pushes in the simulator and the real Create as described in the User Guide on page 7.
`ButtonAdv` – Double of Boolean value, 1 if the Advance (**>>|**) button is pushed, 0 if not. Note the difference between button pushes in the simulator and the real Create as described in the User Guide on page 7.
`Dist` – Double, meters traveled since last call to `AllSensorsReadRoomba` or `DistanceSensorRoomba`. The value saturates at ± 32.768 m.
`Angle` – Double, radians turned since last call to `AllSensorsReadRoomba` or `AngleSensorRoomba`. The value saturates at ± 571 rad.
`Volts` – Double, voltage of the battery in volts. The value should be between 17.2 and 0 V. The simulator will always return a full battery.
`Current` – Double, current through the battery in amps. Positive indicates a charging battery, negative is discharging. The simulator will always output -3 A.
`Temp` – Double, temperature of the battery in Celsius. The simulator always assumes the battery to be at room temperature (25°C).
`Charge` – Double, remaining charge in the battery in milliamp-hours. The simulator always assumes a fully charged battery at 3000 mAh.
`Capacity` – Double, totally possible charge in the battery in milliamp-hours. The simulator always assumes a new battery at 3000 mAh.
`pCharge` – Double, charge remaining in the battery as a percentage of the capacity. The simulator always assumes a fully charged battery at 100%.

AngleSensorRoomba

```
AngleR= AngleSensorRoomba(obj)
```

Summary

The function reads the angular odometry value stored in the robot. It will return the angle that the robot has turned since the last call to this function or `AllSensorsReadRoomba`. The odometry can store no angle larger than ± 571 radians, so the odometry must be read at frequent intervals to avoid saturation.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`Angle` – Double, radians turned since last call to `AllSensorsReadRoomba` or `AngleSensorRoomba`. The value saturates at ± 571 rad.

BatteryChargeReaderRoomba

```
[Charge Capacity Percent]= BatteryChargeReaderRoomba(obj)
```

Summary

On the real Create, this function reads several charge-related values from the battery. This may be used to end autonomous code execution if the battery is low. In the simulator, the values will be constant no matter when the function is called.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`Charge` – Double, remaining charge in the battery in milliamp-hours. The simulator always assumes a fully charged battery at 3000 mAh.
`Capacity` – Double, total possible charge in the battery in milliamp-hours. The simulator always assumes a new battery at 3000 mAh.
`Percent` – Double, charge remaining in the battery as a percentage of the capacity. The simulator always assumes a fully charged battery at 100%.

BatteryVoltageRoomba

```
Voltage= BatteryVoltageRoomba(obj)
```

Summary

On the real Create, this function returns the voltage level of the battery. The simulator will always assume a perfect battery.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`Voltage` – Double, voltage of the battery in volts. The value should be between 17.2 and 0 V. The simulator will always return a full battery.

BeepRoomba

```
BeepRoomba(obj)
```

Summary

This function will cause the Create to beep. The simulator uses the MATLAB `beep` function that will play the computer's default beep (can sound like an error message pop-up). Speakers must be turned on to hear this for the simulator.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

BumpsWheelDropsSensorsRoomba

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

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

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

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.

ButtonsSensorRoomba

```
[ButtonAdv ButtonPlay]= ButtonsSensorRoomba(obj)
```

Summary

This function will read the state of the Play (**>**) and Advance (**>>|**) buttons on the robot. On the real Create, this function will only indicate a button is active if it is being held down. In the simulator, the button functionality is provided by toggle buttons. Read more about the difference in the User Guide on page 7.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`ButtonAdv` – Double of Boolean value, 1 if the Advance button is active, 0 if not.
`ButtonPlay` – Double of Boolean value, 1 if the Play button is active, 0 if not.

CameraSensorCreate

```
[angle dist color]= CameraSensorCreate(obj)
```

Summary

This function will return the output from a generic blob-detection algorithm that processes the camera's view. It is assumed that the blob-detection algorithm is looking for circles of various specific colors. The simulator also assumes that that camera detects beacons perfectly up to the range of the camera, then no further. Empty vector output indicates that no beacons were seen.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`angle` – Vertical array of doubles, each entry being the angular position in radians of the observed beacon relative to the forward heading of the robot. Positive values indicate the beacon is to the left of the front of the robot, negative values indicate the right.
`dist` – Vertical array of doubles, each entry being the distance in meters from the camera to the observed beacon.
`color` – Matrix of doubles (3 columns wide), each row being the color vector of the beacon.

CliffFrontLeftSensorRoomba

```
state= CliffFrontLeftSensorRoomba(obj)
```

Summary

This function will sense for a cliff beneath the front-left cliff sensor. There are no cliffs in the simulator.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`state` – Double of Boolean value, 1 if there is a cliff, 0 if not. The simulator always returns 0.

CliffFrontLeftSignalStrengthRoomba

```
strg= CliffFrontLeftSignalStrengthRoomba(obj)
```

Summary

On the real Create, this function will read the signal strength of the front-left cliff sensor. This value is proportional to the reflectivity of the ground, which usually correlates to the color of the ground. In the simulator, the normal ground is considered to be white, while lines are considered black. The cliff sensor signal strength will be much lower over lines than over the ground.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`strg` – Double, signal strength of the cliff sensor as a percentage of the maximum.

CliffFrontRightSensorRoomba

```
state= CliffFrontRightSensorRoomba(obj)
```

Summary

This function will sense for a cliff beneath the front-right cliff sensor. There are no cliffs in the simulator.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`state` – Double of Boolean value, 1 if there is a cliff, 0 if not. The simulator always returns 0.

CliffFrontRightSignalStrengthRoomba

```
strg= CliffFrontRightSignalStrengthRoomba(obj)
```

Summary

On the real Create, this function will read the signal strength of the front-right cliff sensor. This value is proportional to the reflectivity of the ground, which usually correlates to the color of the ground. In the simulator, the normal ground is considered to be white, while lines are considered black. The cliff sensor signal strength will be much lower over lines than over the ground.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`strg` – Double, signal strength of the cliff sensor as a percentage of the maximum.

CliffLeftSensorRoomba

```
state= CliffLeftSensorRoomba(obj)
```

Summary

This function will sense for a cliff beneath the left cliff sensor. There are no cliffs in the simulator.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`state` – Double of Boolean value, 1 if there is a cliff, 0 if not. The simulator always returns 0.

CliffLeftSignalStrengthRoomba

```
strg= CliffLeftSignalStrengthRoomba(obj)
```

Summary

On the real Create, this function will read the signal strength of the left cliff sensor. This value is proportional to the reflectivity of the ground, which usually correlates to the color of the ground. In the simulator, the normal ground is considered to be white, while lines are considered black. The cliff sensor signal strength will be much lower over lines than over the ground.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`strg` – Double, signal strength of the cliff sensor as a percentage of the maximum.

CliffRightSensorRoomba

```
state= CliffRightSensorRoomba(obj)
```

Summary

This function will sense for a cliff beneath the right cliff sensor. There are no cliffs in the simulator.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`state` – Double of Boolean value, 1 if there is a cliff, 0 if not. The simulator always returns 0.

CliffRightSignalStrengthRoomba

```
strg= CliffRightSignalStrengthRoomba(obj)
```

Summary

On the real Create, this function will read the signal strength of the right cliff sensor. This value is proportional to the reflectivity of the ground, which usually correlates to the color of the ground. In the simulator, the normal ground is considered to be white, while lines are considered black. The cliff sensor signal strength will be much lower over lines than over the ground.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`strg` – Double, signal strength of the cliff sensor as a percentage of the maximum.

CurrentTesterRoomba

```
Current= CurrentTesterRoomba(obj)
```

Summary

On the real Create, this function reads the current flowing in or out of the battery. A positive value indicates the battery is charging. A negative value indicates the battery is discharging (the robot is in use). The simulator will always give the same value for a discharging battery.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`Current` – Double, current through the battery in amps. The simulator will always output -3 A.

DemoCmdsCreate

```
DemoCmdsCreate(obj,DemoNum)
```

Summary

On the real Create this function will display a variety of behaviors based on the input. The simulator does not yet have the demo functionality, so it will display a message instead.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

`DemoNum` – Double, number designating the demo to run. Must be an integer between -1 and 9.

-1 - Abort current demo
0 - Cover a room using a combination of behaviors
1 - Cover a room unless signal from Home Base is found
Note: Home base will not be found in the simulator
2 - Spirals outward then inward
3 - Searches for a wall, then drives along the wall
4 - Drives in a figure-8
5 - Drives forward when pushed while avoiding obstacles
Robot cannot be pushed in the simulator
6 - Drives towards a virtual wall
7 - Bounces between 2 virtual walls
8 - Plays Pachelbel's Cannon when cliff sensors are triggered
9 - Plays a different note for each cliff and bump sensor

DistanceSensorRoomba

```
Distance= DistanceSensorRoomba(obj)
```

Summary

This will return the odometry reading of the distance the Create traveled since the last call to `DistanceSensorRoomba` or `AllSensorsReadRoomba`. After calling either of those functions, the 'distance sensor' will reset.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`Distance` – Double, distance in meters since last call to function. Positive distance indicates forward movement, while negative indicates backward. The 'sensor' will saturate at ±32.768 m.

LidarSensorCreate

```
distScan= LidarSensorCreate(obj)
```

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

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`distScan` – Array of doubles, each entry corresponding to the distance in meters from the LIDAR to the nearest obstacle at a certain angle, or to the range of the LIDAR if no obstacle is close enough. Readings will be returned as the minimum range of the LIDAR if the distance to an obstacle is below the minimum detectable distance. The array will have the number of entries based on #entries = angularRange/angularResolution.

OverheadLocalizationCreate

```
[x y th]= OverheadLocalizationCreate(obj)
```

Summary

This function returns the position and orientation of the Create in global coordinates, as seen from the overhead localization system. The simulator is designed for a system of cameras and 2-dimensional barcodes, but any system with a high degree of accuracy should work.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`x` – Horizontal position of the center of the robot in meters. Uses the Cartesian coordinate system.
`y` – Vertical position of the center of the robot in meters. Uses the Cartesian coordinate system.
`th` – Orientation of the robot (angle the robot is facing) in radians. Values are relative to the positive x-axis, and positive counter-clockwise, negative clockwise, between –π and π rad.

ReadBeacon

```
[X,Y,Z,ROT,Ntag]= ReadBeacon(obj)
```

Summary

This function returns the position and orientation in camera coordinates, as well as AR Tag Number of each beacon within the field-of-view of the sensor.

Camera coordinates are defined as:

X: Points to the robots left, parallel to the positive robot Y-axis
Y: Points up (is set to zero for all AR tags, since the simulation is planar)
Z: Parallel to the positive robot X-axis

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`X` – Vector of x-coordinates of each beacon in the camera field-of-view as seen from camera coords.
`Y` – Vector of y-coordinates of each beacon in the camera field-of-view as seen from camera coords.

`Z` – Vector of z-coordinates of each beacon in the camera field-of-view as seen from camera coords.
`ROT` – Vector of orientation values of each beacon in the camera field-of-view. Orientation is assumed zero for all beacons in simulation.
`Ntag` – Vector of beacon numbers for each beacon in the camera field-of-view

ReadSonar

```
distance= ReadSonar(obj)
distance= ReadSonar(obj,sonarNum)
```

Summary

This function returns the distance reading from the front sonar sensor. It is based off of the WOOSH board. Note that the functionality has been changed significantly from the original function.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.
`sonarNum` - Double, number specifying which sonar is to be read. Must be an integer between 1 and 4. 1 corresponds to the right sonar, 2 – front, 3 – left, 4 – back. This input argument is optional. Note that the original

Output

`distance` – Double, distance to the nearest wall in front of the robot, or the range of the sonar if no wall is within range.

ReadSonarMultiple

```
distance= ReadSonarMultiple(obj,sonarNum)
```

Summary

This function returns the distance reading from the specified sonar sensor. It is designed to be compatible with the WOOSH board and the latest firmware. The simulator assumes that the sonar sensors are connected in a certain order to the WOOSH board (see sonarNum specification).

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.
`sonarNum` – Double, number specifying which sonar is to be read. Must be an integer between 1 and 4. 1 corresponds to the right sonar, 2 – front, 3 – left, 4 – back.

Output

`distance` – Double, distance to the nearest wall in front of the robot, or the range of the sonar if no wall is within range.

RealSenseDist

```
depth_array = RealSenseDist(obj)
```

Summary

This function returns an array of Double, in which each entry represents the depth measured from the camera do the obstacle.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`depth_array` – delay and depth measurements. [10 x 1] matrix of floats. depth_array(1) = delay in reading, depth_array(2:10) = depth to point. depth_array(2) is the left most point, depth_array(10) is the right most point. Depth measurements are in meters

RealSenseTag

```
tags = RealSenseTag(obj)
```

Summary

This function returns a 2D array representing the apriltags seen in the RGB image as well as their location. If no tags are detected, then an empty array is returned.

Camera coordinates are defined as:

X: Points to the robots left, parallel to the positive robot Y-axis
Y: Points up (is set to zero for all AR tags, since the simulation is planar)
Z: Parallel to the positive robot X-axis

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`tags` – 2D array, represents a tag location in this order: [dt id z x yaw].
- "dt" is the delay in the reading
- "id" is the apriltag ID.
- "z" is the distance between the center of the tag and the camera, in the camera frame along the axis perpendicular to the camera.
- "x" is the distance between the center of the tag and the center of the image, in the camera frame along the horizontal axis across the image.
- "yaw" is the angle of rotation of the tag about the z axis of the camera frame (axis perpendicular to the camera), in radians.

RoombaInit

```
obj= RoombaInit(obj)
```

Summary

On the real Create this function establishes the serial port connection on the inputted communication port number then starts the robot and confirms connection. In the simulator this function has no purpose since the robot object is already created. It will simply output the same object that is inputted.
It is recommended that the autonomous control program does not include the function `RoombaInit` in it. It would avoid unnecessary opening and closing of the serial port if `RoombaInit` is called prior to running the real Create, and the serial port object is passed into the autonomous control program in the input argument. This will work the same way in the simulator (although `RoombaInit` does not need to be called prior to running the simulator).

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`obj` – User-defined object of class `CreateRobot`, should be used as the first input argument to all subsequent functions.

SetDriveWheelsCreate

```
SetDriveWheelsCreate(obj,rightWheel,leftWheel)
```

Summary

This command will set the linear velocity of each of the drive wheels individually. To set the angular velocity of the wheels the radius of the wheels must be known. To set the total linear and angular velocities of the Create with this function the distance between the drive wheels must be known.

Note that the velocity limits on each wheel expressed here apply during the execution of other functions as well. Thus, for functions such as `SetFwdVelRadiusRoomba` or `SetFwdVelAngVel` movement at the maximum forward and also maximum angular velocity is impossible. The velocity combination must be set such that each wheel does not exceed the limits. Otherwise the function will automatically limit the wheel speeds, which may change the curvature of the robot's path.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.
`rightWheel` – Linear velocity of the right wheel in meters per second. Positive values indicate forward movement, negative indicates backwards. Values must be between -0.5 and 0.5 m/s.
`leftWheel` – Linear velocity of the left wheel in meters per second. Positive values indicate forward movement, negative indicates backwards. Values must be between -0.5 and 0.5 m/s.

SetFwdVelAngVelCreate

`SetFwdVelAngVel(obj,FwdVel,AngVel)`

Summary

This command will set the overall forward and angular velocities of the Create. This is probably the most easily controlled movement command.

See the note in the summary of `SetDriveWheelsCreate` for information on wheel limits.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.
`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.

SetFwdVelRadiusRoomba

`SetFwdVelRadiusRoomba(obj,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.

See the note in the summary of `SetDriveWheelsCreate` for information on wheel limits.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.
`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

SetLEDsRoomba

`SetLEDsRoomba(obj,LED,pColor,pIntensity)`

Summary

This will change the settings for the three LEDs on the button panel of the Create. The LEDs next to the Play (>) and Advance (>>|) buttons will be referred to as Play and Advance. These two LEDs can be switched on and off, while the LED next to the power button can be a range of shades between green and red, and a range of brightness settings.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.
`LED` – Double, controls the state of the Play and Advance LEDs. Must be an integer between 0 and 3.

- 0 – Both off
- 1 – Advance on, Play off
- 2 – Advance off, Play on
- 3 – Both on

`pColor` – Double, controls the color of the power LED. Must be between 0 and 100. 0 is pure green, 100 is pure red, numbers scale in between.
`pIntensity` – Double, controls the brightness of the power LED. Must be between 1 and 100. 1 is very dim, 100 is very bright, numbers scale in between. Think percentage of intensity.

travelDist

`travelDist(obj,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

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.
`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(obj,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

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.
`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.

VirtualWallSensorCreate

```
state= VirtualWallSensorCreate(obj)
```

Summary

This function checks if the robot's IR sensor is detecting a virtual wall. This requires a direct line of sight between the virtual wall emitters and the Create's IR sensor.

Input

`obj` – User-defined object of class `CreateRobot`, the same object that is the input to the autonomous control program.

Output

`state` – Double of Boolean value, 1 if a virtual is detected, 0 if not.

Contact Information

Suggestions and bugs can be reported in the appropriate forum on the project's SourceForge page:
Forums: https://sourceforge.net/projects/createsim/forums/forum/1204154

Other inquiries can be directed to:
Email: CreateMatlabSim@gmail.com

For more information, visit:
Website: http://web.mae.cornell.edu/hadaskg/CreateMATLABsimulator/createsimulator.html
SourceForge: https://sourceforge.net/projects/createsim

Contributing Parties

Author:
Cameron Salzberger

Advising Professors:
Dr. Hadas Kress-Gazit
Dr. K-Y Daisy Fan

Coding and Testing Assistance:
Jason Hardy
Francis Havlak
Ankit Arora

Creaters of the MATLAB Toolbox for the iRobot Create:
Joel M. Esposito
Owen Barton
http://www.usna.edu/Users/weapsys/esposito/roomba.matlab/
2008

Code Resources:
Nassim Khaled (inside_triangle on MATLAB Central)
Zhenhai Wang (circle on MATLAB Central)

Sponsor:

The MathWorks
http://www.mathworks.com/