4
Reference

4.1 Introduction

This section provides details on all of the functions and commands related to the Khepera Toolbox. The reference tables in the next section are broken up into functional tables. Interface Functions are the core functions and provide Matlab command prompt access to a real or simulated Khepera. Other function groupings include Graphical User Interfaces, Support Functions (for Plotting and for the Simulator), and a Utility Library. This last grouping is just a cluster of functions found useful by the author but which have no integral use by the Toolbox.

4.2 Quick Reference Tables

Interface Functions: REAL KHEPERA	
con	Configure parameters of the speed controller
skp	Set a position to be reached
sms	Set the speed of the motors
rms	Read the instantaneous motor speed
cpid	Configure the position PID controller
spc	Set the position counter
rpc	Read the position counter
rad	Read A/D input
cspc	Configure the speed profile controller
rmc	Read the status of the motion controller
cls	Change LED state
rps	Read proximity sensors
rls	Read ambient light sensors

Interface Functions: SIMULATED KHEPERA

Interface Functions: SIMULATED KHEPERA	
sim_skp	Set a position to be reached
sim_sms	Set the speed of the motors
sim_rms	Read the instantaneous motor speed
sim_spc	Set the position counter
sim_rpc	Read the position counter
sim_rmc	Read the status of the motion controller
sim_rps	Read proximity sensors
sim_rls	Read ambient light sensors

Graphical User Interfaces (GUI's)	
khepbase	Displays the Khepera sensors in bargraph format
play_pen	An editable GUI for creating and saving simulation environments
roborun2	A GUI for automated algorithm iteration
simsense	A GUI for measuring sensor values on a simulated khepera
tstsense	A GUI for measuring sensor values on a real khepera
template	A template m-file to use with roborun2

Support Functions: PLOTTING	
datameta	Print the current graph to the Windows Clipboard
dataprnt	Print the current graph to the printer
ddl	Displays data collected by all sensors during a linear trajectory
ddlg	Displays data as in ddl but with a grid
ddr	Displays data collected by all sensors during a rotational trajectory
ddrg	Displays data as in ddr but with a grid
disonel	Displays data for a single sensor obtained during linear or rotational trajectory

Support Functions: SIMULATOR	
checkobj	Checks to see if an “object” is in front of a simulated Khepera
kheprom	Emulates the functionality of the ROM monitor in the real Khepera
movekhep	Translates a simulated Khepera along a linear trajectory
putkhep	Places a simulated Khepera at a point in a simulated environment as indicated by the user
rotkhep	Rotates a simulated Khepera
upsv	Updates the proximity sensor values of a simulated Khepera in a simulated environment
ulsv	Updates the light sensor values of the simulated Khepera in a simulated environment
cvrtang	Converts angles to under 180 degrees

Utility Library

Utility Library	
user1	Implements a GUI for printing a saving statistics data collected during a run of an algorithm
getstat	Tracks statistics for an algorithm
plotrun	Plots the statistics collected by getstat

4.3 Commands and Functions

The reference section on the following pages describes in detail the functionality of each command listed in the tables above.

checkobj

Purpose

Checks for the presence of an object in front or behind the simulated khepera.

Synopsis

```
[obj_in_front,obj_in_back]=checkobj(btmap1)
```

Description

This function is used in the simulation environment to check for the presence of an object around the perimeter of the simulated khepera.

This function receives `btmap1`, a 500X600 matrix whose elements represent the colourmap index of each pixel in the simulated environment presented within the current **Khepera Virtual Playpen** window. It returns a binary indication of objects in front or in back (1=object, 0=no object) through the return variables `obj_in_front` and `obj_in_back`.

Notes:

- This function achieves its means by defining a front and back sweep vector of angles in the range [-50,50] in 5 degree increments. These angles are used to produce a vector of X and Y indices into the `btmap1` variable in the following manner:

  - ! add each angle increment to the absolute angle of the khepera
  - ! take the cosine of the resultant angle
  - ! multiply the cosine by 29, the radius (in mm) of the khepera plus one
  - ! add the khepera's absolute X and Y position to the X and Y values computed above

- These computations result in the X and Y values describing a curve that follows just outside the perimeter of the khepera both in front and in back. These values are used as indices into the `btmap1` variable to determine the colourmap entry at each point along the curve. If the colourmap entry at any point is 44 then there is an object at that point.

- To increase the granularity of the check, reduce the increment at the indicated locations in the m-file. To widen the sweep, increase

the range boundaries in the m-file. Take note that widening the sweep angle in front and in back risks causing the khepera to "stick" to walls. That is, the khepera should logically be able to proceed parallel to a wall. However, if the front sweep angle is 89 degrees and the khepera is very close to the wall, then an object will be detected and forward motion will be inhibited even though nothing appears to be impeding the motion.

See Also `movekhep()` `checkobj.m`

References Khepera User Manual, Version 4.06

cls

Purpose Change LED state

Synopsis `cls(port_id,led_number,action_number)`

Description `cls` changes the status of the two LED's on top of the Khepera through the serial port indicated in `port_id`. Valid values for port_id are 1-4. The LED's are numbered zero and 1 and `action_number` can take the values 0 (turn OFF), 1 (turn ON), 2 (toggle).

References Khepera User Manual, Version 4.06

con

Purpose Configure the parameters of the speed Khepera speed controller

Synopsis `con(port_id,Kp,Ki,Kd)`

Description `con` sets the proportional (Kp), integral (Ki) and derivative (Kd) parameters of the speed controller of a real Khepera through the serial port defined in port_id. Valid serial ports are 1-4.

See Also `cspc,cpid`

References Khepera User Manual, Version 4.06

cpid

Purpose Configure the position PID controller

Synopsis `cpid(port_id,Kp,Ki,Kd)`

Description `cpid` sets the proportional (Kp), integral (Ki), and derivative (Kd) parameters of the position regulator thought the serial port indicated in `port_id`. Valid values for port_id are 1-4. At reset, Kp=3000, Ki=20, and Kd=4000.

See Also `con,skp,spc,rpc`

References Khepera User Manual, Version 4.06

cspc

Purpose Configure the speed profile controller

Synopsis

```
cpsc(port_id,max_speed_left,acc_left,
     max_speed_right,acc_right)
```

Description `cpsc` sets the speed and acceleration for the trapezoidal speed shape of the position controller through the serial port indicated in `port_id`. Valid values for `port_id` are in the range [1,4]. The `max_speed` parameter indicates the maximum speed attained during displacement and has units pulse/10ms (or 0.08mm/s). The `acc` parameter specifies the acceleration to be applied to each motor during the displacement and has units of ((pulse/256)/10ms^2) which corresponds to 3.125 mm/s^2. At reset, these parameters are max_speed=20 and acc=64.

See Also `skp,spc,rpc`

References Khepera User Manual, Version 4.06

datameta

Purpose Copies the current figure window to the Windows clipboard.

Synopsis `datameta()`

Description `datameta` is a low level function used to copy the contents of the topmost figure window with the Matlab tag *graph_data* to theWindows clipboard.

dataprnt

Purpose Prints the current figure window.

Synopsis `dataprnt()`

Description `dataprnt` is a low level function used to print the contents of the topmost figure window with the Matlab tag *graph_data.*

Notes:

- Modify the **print** command in `dataprnt.m` m-file to match the desired printer. Currently, this file executes **print -dbj10e** which formats the data for the *Cannon Bj10e* series printers.

ddl

Purpose Display linear data for all sensors in one graph without grids.

Synopsis `ddl(dv,khep_pos)`

Description `ddl` is a low level function used by `tstsense.m` and `simsense.m` for the production of a figure containing eight graphs, one for each sensor. This function provides the graphs (without a grid) for data taken from a **linear** test trajectory. `dv` is a matrix of test values taken during a test run on either a simulated khepera or on a real khepera. Fields in `dv` are arranged as follows:

Columns 1-2: Khep position counter values at each measurement
Columns 3-10: Measured values of light or proximity (depending on test performed), one column per sensor.

`khep_pos` is the maximum rotational or linear distance travelled during the last test (in pulses).

See Also `testsense.m, simsense.m`

ddlg

Purpose Display linear data for all sensors in one graph with a grid on each graph.

Synopsis `ddlg(dv,khep_pos)`

Description `ddlg` is a low level function used by `tstsense.m` and `simsense.m` for the production of a figure containing eight graphs, one for each sensor. This function provides the graphs (with a grid) for data taken from a **linear** test trajectory. `dv` is a matrix of test values taken during a test run on either a simulated khepera or on a real khepera. Fields in `dv` are arranged as follows:

Columns 1-2: Khep position counter values at each measurement
Columns 3-10: Measured values of light or proximity (depending on test performed), one column per sensor.

`khep_pos` is the maximum rotational or linear distance travelled during the last test (in pulses).

See Also `testsense.m, simsense.m`

ddr

Purpose Display rotational data for all sensors in one graph without grids.

Synopsis `ddr(dv,khep_pos)`

Description `ddr` is a low level function used by `tstsense.m` and `simsense.m` for the production of a figure containing eight graphs, one for each sensor. This function provides the graphs (without a grid) for data taken from a **rotational** test trajectory. `dv` is a matrix of test values taken during a test run on either a simulated khepera or on a real khepera. Fields in `dv` are arranged as follows:

Columns 1-2: Khep position counter values at each measurement
Columns 3-10: Measured values of light or proximity (depending on test performed), one column per sensor.

`khep_pos` is the maximum rotational or rotational distance travelled during the last test (in pulses).

See Also `testsense.m, simsense.m`

ddrg

Purpose

Display rotational data for all sensors in one graph with a grid on each graph.

Synopsis

```
ddrg(dv,khep_pos)
```

Description

`ddrg` is a low level function used by `tstsense.m` and `simsense.m` for the production of a figure containing eight graphs, one for each sensor. This function provides the graphs (with a grid) for data taken from a **rotational** test trajectory. `dv` is a matrix of test values taken during a test run on either a simulated khepera or on a real khepera. Fields in `dv` are arranged as follows:

Columns 1-2: Khep position counter values at each measurement
Columns 3-10: Measured values of light or proximity (depending on test performed), one column per sensor.

`khep_pos` is the maximum rotational or rotational distance travelled during the last test (in pulses).

See Also

`testsense.m, simsense.m`

disonel

Purpose Display data for a single sensor

Synopsis `disonel(dv,khep_pos,which_one)`

Description `disonel` is a low level function used by `tstsense.m` and `simsense.m` for the production of a figure containing data for one sensor. `dv` is a matrix of test values taken during a test run on either a simulated khepera or on a real khepera. Fields in `dv` are arranged as follows:

Columns 1-2: Khep position counter values at each measurement
Columns 3-10: Measured values of light or proximity (Depending on test performed), one column per sensor.

`khep_pos` is the maximum rotational or rotational distance travelled during the last test (in pulses).

The column of `dv` containing the data to be displayed is selected by the variable `which_one` which is passed from the `simsense` or `testsense` gui.

See Also `testsense.m, simsense.m`

khepbase

Purpose

Create a visual display for the Khepera Base unit in operation

Synopsis

```
khepbase
khepbase(prox_data,light_data,motor_speed)
```

Description

`khepbase` is a Read-Only graphical user interface that provides an informative picture of what the proximity and light sensors are reading. This is done by a pair of bar graphs stationed at each sensor position in an overhead view of the khepera. Additionally, a bargraph is placed beside each motor to indicate the current speed. `khepbase` must be called once without arguments before any other calls to this function are made. This call draws the interface and initializes variables. Each successive call must provide each of `prox_data` - a 1! 8 vector of proximity values (between 0 and 1024), `light_data` - a 1! 8 vector of light data (between 0 and 500), and `motor_speed` - a 1! 2 vector of motor speeds (between 0 and 20).

Usage Example:

The following example shows an appropriate pair of calls to `khepbase` in the `brait.m` function (see section 2). The first initializes the interface, and the second provides information gathered from calls to a real khepera.

```
%BRAIT      Braitenburg Vehicle Simulation
%
khepbase;
loopvar=0;
Intercon=[4 -5;4 -15;6 -18; -18 6; -15 4; -5 4;5 3;3 5];
speed=[10 10];
while loopvar == 0
      motors=(rps(2)*Intercon)/400+speed;
      sms(2,motors(1),motors(2));
      ld=rls(2); %Get light data
      pd=rls(2); %Get proximity data;
      khepbase(pd,ld,speed); %Update the display
end
```

References

Khepera User Manual, Version 4.06

kheprom

Purpose Emulates the functionality of the ROM in a real Khepera

Synopsis `kheprom(btmap1)`

Description `kheprom` isolates the higher level serial communications functions (`sim_rps, sim_sms,` etc) from the code required to implement the instructions on a simulated khepera.

`btmap1` is the MAT file variable containing the image map of the simulator environment. This variable is created using the function `play_pen()` and is stored in a MAT file of the user's choosing. A default environment is provided in the btmap1 variable of the MAT file 'defmap'.

The interaction between the function kheprom() and the simulator serial functions is shown in the diagram below. Basically, the simulated serial communications functions act to modify the contents of various registers (described below). `kheprom()` operates on these registers in the same manner as the ROM on board the Khepera by interpreting their contents and carrying out the required actions.

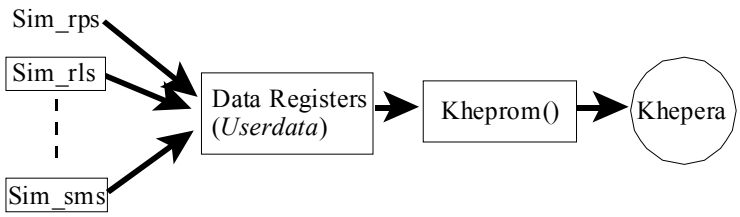

Diagram 3-1
Simulated Serial Command Interaction With Kheprom()

In the simulation environment, a Matlab colour patch object is created to represent the Khepera. All graphics objects in Matlab have a *userdata* property (which allows the storage of a vector or matrix within the object) and a *tag* property which can be used to assign each graphics object a unique name for identification purposes. Knowing and preserving the patch object's *tag* gives access to all of the properties of the object,

including the *userdata* property. In this case, the data stored in the Khepera patch object's *userdata* area is a vector whose fields represent both the real registers on the Khepera and some general purpose simulator-related registers (defined as required). This vector is defined as follows:

Columns 1-3:	absolute x, y, and rotation of khep in degrees
Columns 4-11:	proximity sensor data
Columns 12-19:	light sensor data
Columns 20-21	Speed motor left, Speed motor right
Columns 22-27	Motion control registers.Left: T,M,E Right: T,M,E
Columns 28-29	Position counter registers
Columns 30-31	Position controller registers - position to be reached.

So, for a function like `sim_rps` to return proximity data when called, it simply finds the graphics object whose tag is "Khep" and reads the userdata vector, positions 4 to 11. To set the motor speeds, `sim_sms` will find the "Khep" graphics object, write a '1' to the mode bits of the userdata vector (23 and 26), and then write the desired speeds into the speed fields (20-21). When `kheprom()` is next executed, all fields are updated and actions are performed according to the contents of the vector in *userdata*.

Notes:

- the above description implies that in an iterative algorithm, `kheprom()` must be run once *every* cycle in order to update the disposition of the simulated Kheprom patch object.

- This version of the software allows only one Khepera in the environment at a time. If there were more than one, the command to find the object with the tag "Khep" would return a vector of handles to all the objects so named.

- The `kheprom.m` m-file has two variables, *rot_const* and *gran*, which set the granularity of the simulated rotation and translation. Initially, these are 15 degrees per iteration of the `rotkhep` command and 5 pixels (mm) per iteration of the `movekhep` command. To change the granularity, these variables must be modified in this m-file (`kheprom.m`) and, additionally, the *gran* variable in the m-file `movekhep.m` must also be modified. There is **no** *rot_const* variable in the rotkhep.m m-file since the granularity is passed to this function as an argument.

- `kheprom.m` emulates both speed and position modes of operation

of the khepera (see the **C**, **D**, and **G** commands in the *Khepera User Manual* for a description of these modes of operation). It does this as the real khepera would - by checking the status of the motion control bits of the *userdata* vector (bits 22-27). In actuality, only bit 23 is checked to determine the mode of operation since both motors are assumed to always be in the same mode (see the **K** command in the *Khepera User Manual* for a description of motion control status registers). In the simulation, the error bits (24, 27) are never set. If the mode bit is zero (speed mode), then the following actions are performed:

- If the `speed_motor_left` register (position 20 of the *userdata* vector) and `speed_motor_right` register (21) both contain numbers greater than zero, then the desired motion is forward (see `sim_sms.m` for the setting of these registers). The `movekhep` command is then issued with direction 1 (see `movekhep.m`).

- If the `speed_motor_left` and right registers both contain numbers less than zero, then the desired motion is reverse. The `movekhep` command is then issued with direction -1 (see `movekhep.m`).

- If the `speed_motor_left` register is less than the `speed_motor_right` register, then the desired motion is a rotation to the left. The `rotkhep` command is issued with a negative rotation constant (see `rotkhep.m`).

- If the `speed_motor_left` register is greater than the `speed_motor_right` register, then the desired motion is a rotation to the right. The `rotkhep` command is issued with a positive rotation constant (see `rotkhep.m`).

The simulated khepera can therefore only emulate four types of motion - forward, backward, rotate left, and rotate right. The magnitude of the numbers in the speed registers are irrelevant as long as they correctly reflect one of these types of motion. The corollary to this simplification is that the simulator will NOT emulate the operation of the Khepera with different speeds in each motor.

- If the mode bit is set to 1 (position mode), then the following actions are performed by kheprom.m:

- Each time `kheprom.m` is iterated, the `Target` bit for the left motor (bit 22 of the *userdata* vector) is checked to see if the simulated khepera is at the desired location. If yes, no action is performed.

- If the simulated khepera is not at the desired location, then the position control registers (30 and 31 of *userdata*) and the position counter registers (28 and 29 of *userdata*) are compared. If both position control registers and both position counter registers have the same sign, then the simulated khepera is commanded to move along linear trajectory. Otherwise, a rotational motion is assumed.

- The units of the position counter and position control registers are pulses (see the **C** command in the *Khepera User Manual*). For linear motion, there are 12.5 pulses per millimetre (0.08 mm/p) and therefore 12.5*! gran* pulses in one iteration of the `movekhep` command (where gran is the granularity of motion defined in `kheprom.m` and `movekhep.m`). For rotational motion, there are 2πr! 12.5 pulses per millimetre (where r=26mm is the radius from the geometric centre of a real khepera to the centre of an outside wheel) and therefore 2042! *rot_const*/360 pulses in one iteration of the `rotkhep` command. The difference between the position counter register (left motor) and the position control register (left motor) is computed. If the computed difference is less than the number of pulses in one iteration of either linear or rotational motion (as computed above), then the position is deemed to have been reached and the `Target` bits (22 and 25) are set. If not, either `movekhep` or `rotkhep` are iterated depending on the type of motion chosen.

- As implied from the notes above, this version of the simulator does not allow the mixing of rotational and linear motion to achieve an arbitrary position as does the real khepera. All target positions provided by the user will result in either linear or rotational motion, but not both. Hence, when a series of target positions are fed to the simulated khepera, the position counters will accurately reflect the position if all the targets resulted in linear or rotational motion. For example, if a target position was specified and the khepera moved to that position along a linear path, the position counters

would be then reflect an accurate position count. If a second position was specified and it also resulted in a linear motion, then the position counters would still be accurate. However, if a third position resulted in a rotation, then the integrity and indeed the behaviour of the khepera cannot be guaranteed. This problem can be overcome by setting the position control registers to zero (eg `sim_skp(1,0,0)`) prior to setting any new target position.

See Also `sim_skp, rotkhep.m, movekhep.m, sim_rps, sim_sms, play_pen()`

References Khepera User Manual, Version 4.06

movekhep

Purpose Moves the simulated khepera in the Virtual Playpen

Synopsis `movekhep(dirn,btmap1)`

Description `movekhep` is used in the simulation environment to move the khepera image object in the Virtual Playpen environment.

This function receives `btmap1` but does not use it directly. Rather, it is passed on to checkobj when called by this function. The variable `dirn` indicates the direction of motion to move. `movekhep` moves the simulated khepera in a linear fashion only. Thus, `dirn` takes the values -1 for reverse, 1 for forward.

Notes:

- This function moves the simulated Khepera exactly `gran` pixels, where gran is the granularity variable. To move more pixels per iteration, increase `gran`. When modifying gran in this file, ensure that it is also modified in `kheprom`. Increasing the value of `gran` will speed up the simulation.
- This function also updates the position counter of the simulated khepera. The maximum value of this counter is limited to 2147483648 (the counter is 32 bits and the most significant bit is assumed to be the sign bit). Even if the khepera runs into a wall, the position counter values will continue to be updated. This reflects reality since the wheels of the robot will spin when an object is encountered.
- For speed, the patch object is not re-rendered. Instead, its x-and-ydata are read, modified, and written back, which makes for much smoother animation.

See Also `rotkhep,kheprom`

References Khepera User Manual, Version 4.06

play_pen

Purpose

Open an interactive environment for the creation of "Virtual Playpens" or environments for the simulated khepera to navigate.

Synopsis

```
play_pen
```

Description

`play_pen` provides a simple, interactive graphical user interface (gui) for creating different environments for a simulated khepera robot to navigate. See the tutorial for a complete description of functionality and use.

Notes:

- The environment that is drawn in the Virtual Playpen window will be inverted when it is scanned and opened from `roborun2.`
- Periodically, the top of the Virtual Playpen will disappear. It can be brought back by placing an object, moving it, then deleting it.

See Also

`roborun2`

References

Khepera User Manual, Version 4.06

putkhep

Purpose

Place the simulated khepera in the Virtual Playpen.

Synopsis

```
putkhep(xabs,yabs)
```

Description

`putkhep` is used in the simulation environment to allow the user to place a simulated khepera in the Virtual Playpen window at the location of his/her choice through the use of a mouse. `putkhep` receives `xabs` and `yabs` which correspond to the desired centre position of the robot in the window.

This function also creates and formats the *userdata* property of the khepera patch object. See `kheprom` or the `putkhep.m` m-file for detailed explanation of the field allocation of the *userdata* vector.

Notes:

- This version does NOT currently incorporate validation of the selected location. Thus, it is possible to place a khepera virtual robot on top of an obstacle (although not recommended).
- Sensors and other niceties are not rendered on the patch object. Instead, an arrowhead is drawn to indicate the current direction of the robot. Sensor rendering may be incorporated later when other functions have been optimized.

See Also

`putkhep.m, kheprom`

References

Khepera User Manual, Version 4.06

rad

Purpose Read an A/D input

Synopsis `[analog_value]=rad(port_id,channel_number)`

Description `rad` reads a 10 bit value thought the serial port indicated in `port_id` corresponding to the analog value in `channel_number`. The maximum value of 1024 is equivalent to an analog value of 4.09 Volts.

References Khepera User Manual, Version 4.06

rls

Purpose Read the ambient light sensors on the Khepera base unit

Synopsis `[light_data]=rls(port_id)`

Description `rls` reads the values of all the ambient light sensors on the base unit through the serial port indicated in `port_id`. Valid values for port_id are 1-4. The `light_data` return variable is a 1X8 vector whose elements correspond to readings of the sensors in the following order: [val_sens_left90, val_sens_left45, val_sens_left10, val_sens_right10, val_sens_right45, val_sens_right90, val_sens_back_right, val_sense_back_left]

See Also `rps`

References Khepera User Manual, Version 4.06

rmc

Purpose Read the status of the Khepera motion controller

Synopsis `[motion_status]=rmc(port_id)`

Description `rmc` reads the status of the motion controller through the serial port indicated in `port_id`. Valid values for port_id are 1-4. The function returns motion_status, a 1X6 vector whose elements correspond to the motion status flags for the left and right motors respectively. The status of each motor is given by three flags: T (target), M (mode), and E (error). T=0 indicates that the robot is still moving towards a target when in position mode. T=1 indicates that the robot has reached a position as commanded by `skp`. M=0 indicates that the current displacement is controlled by the position mode (ie `skp`) whereas M=1 indicates that displacement is being controlled by speed (ie `sms`). E indicates a controller position error. Hence, the motion_status vector is organized as: [T_left, M_left, E_left, T_right, M_right, E_right].

See Also `skp,spc,sms`

References Khepera User Manual, Version 4.06

rms

Purpose Read the instantaneous motor speed of the Khepera

Synopsis `[motor_speeds]=rms(port_id)`

Description `rms` reads the instantaneous speed of the two motors of the khepera through the serial port indicated by port_id. Valid values for port_id are 1-4. The values are returned in `motor_speeds` which is a 1X2 vector whose first element is the left motor speed and second element is the right motor speed. The values in `motor_speeds` have units of pulse/10ms which translates to 8 mm/s.

See Also `sms,cpsc`

References Khepera User Manual, Version 4.06

roborun2

Purpose A graphical user interface for automated algorithm iteration.

Synopsis `roborun2`

Description `roborun2` provides an interactive graphical user interface (gui) for the controlled automation of algorithms. It is also the primary launching point for all other graphical user interfaces. See the overview section for a complete description of functionality and use.

rotkhep

Purpose Rotate the simulated khepera in the Virtual Playpen.

Synopsis `rotkhep(alpha)`

Description `rotkhep` is used in the simulation environment to rotate the khepera image object in the Virtual Playpen environment.

This function receives `alpha`, the rotation angle. To rotate left, `alpha` must be positive and to rotate right, `alpha` must be negative.

Notes:

- This function uses the Matlab **rem** function to restrict the absolute angle maintained by the khepera patch object to the range [0 360]. It uses the Matlab **rotate** function to rotate the khepera patch object. The **rotate** function is only available to Matlab V4.2c and above.
- This function also updates the position counters of the simulated khepera. The maximum value of this counter is limited to 2147483648 (the counter is 32 bits and the most significant bit is assumed to be the sign bit).

See Also `movekhep, kheprom`

References Khepera User Manual, Version 4.06

rpc

Purpose Read the position counters of the Khepera

Synopsis `[position_counters]=rpc(port_id)`

Description `rpc` reads the position counter of each motor through the serial port indicated in `port_id`. Valid values for port_id are 1-4. The function returns `position_counters` which is a 1X2 vector whose first element is the position counter of the left motor and second element is the right motor. The units of the elements in `position_counters` are the pulse, each of which is 0.08mm of physical distance. This equates to 125 pulses per millimetre.

See Also `spc,skp,cpid`

References Khepera User Manual, Version 4.06

rps

Purpose Read the proximity sensors on the Khepera base unit

Synopsis `[prox_data]=rps(port_id)`

Description `rps` reads the values of all the proximity sensors on the base unit through the serial port indicated in `port_id`. Valid values for port_id are 1-4. The `prox_data` return variable is a 1X8 vector whose elements correspond to readings of the sensors in the following order: [val_sens_left90, val_sens_left45, val_sens_left10, val_sens_right10, val_sens_right45, val_sens_right90, val_sens_back_right, val_sense_back_left]

See Also `rls`

References Khepera User Manual, Version 4.06

sim_rls

Purpose Read the ambient light sensors on the simulated Khepera base unit

Synopsis `[light_data]=sim_rls(port_id)`

Description `sim_rls` reads the values of all the ambient light sensors on the simulated base unit. `port_id` is not used in the function but is preserved to maintain compatibility with `rls`. The `light_data` return variable is a 1X8 vector whose elements correspond to readings of the sensors in the following order: [val_sens_left90, val_sens_left45, val_sens_left10, val_sens_right10, val_sens_right45, val_sens_right90, val_sens_back_right, val_sense_back_left]

See Also `sim_rps`

References Khepera User Manual, Version 4.06

sim_rmc

Purpose Read the status of the simulated Khepera's motion controller

Synopsis `[motion_status]=sim_rmc(port_id)`

Description `sim_rmc` reads the status of the motion controller of a simulated Khepera. `port_id` is not used in the function but is preserved to maintain compatibility with `rmc`. The function returns motion_status, a 1X6 vector whose elements correspond to the motion status flags for the left and right motors respectively. The status of each motor is given by three flags: T (target), M (mode), and E (error). T=0 indicates that the robot is still moving towards a target when in position mode. T=1 indicates that the robot has reached a position as commanded by `skp`. M=0 indicates that the current displacement is controlled by the position mode (ie `skp`) whereas M=1 indicates that displacement is being controlled by speed (ie `sms`). E indicates a controller position error. Hence, the motion_status vector is organized as: [T_left, M_left, E_left, T_right, M_right, E_right].

See Also `sim_skp,sim_spc,sim_sms`

References Khepera User Manual, Version 4.06

sim_rms

Purpose Read the instantaneous motor speed of the Khepera

Synopsis `[motor_speeds]=sim_rms(port_id)`

Description `sim_rms` reads the instantaneous speed of the two motors of a simulated khepera. `port_id` is not used in the function but is preserved to maintain compatibility with `rms`. The values are returned in `motor_speeds` which is a 1X2 vector whose first element is the left motor speed and second element is the right motor speed. The values in `motor_speeds` will be those values set by sim_sms. This release of the simulator (V1.0) will pass back motor speeds of zero when `sim_rms` is used during the execution of `sim_skp`.

See Also `sim_sms`

References Khepera User Manual, Version 4.06

sim_rpc

Purpose Read the position counters of a simulated Khepera

Synopsis `[position_counters]=sim_rpc(port_id)`

Description `sim_rpc` reads the position counter of each motor of a simulated Khepera. `port_id` is not used in the function but is preserved to maintain compatibility with `rpc`. The function returns `position_counters` which is a 1X2 vector whose first element is the position counter of the left motor and second element is the right motor. The units of the elements in `position_counters` are the pulse, each of which is 0.08mm of physical distance. This equates to 125 pulses per millimetre.

See Also `sim_spc,sim_skp`

References Khepera User Manual, Version 4.06

sim_rps

Purpose Read the proximity sensors on the simulated Khepera base unit

Synopsis `[prox_data]=sim_rps(port_id)`

Description `sim_rps` reads the values of all the proximity sensors on the simulated base unit. `port_id` is not used in the function but is preserved to maintain compatibility with `rps`. The `prox_data` return variable is a 1X8 vector whose elements correspond to readings of the sensors in the following order: [val_sens_left90, val_sens_left45, val_sens_left10, val_sens_right10, val_sens_right45, val_sens_right90, val_sens_back_right, val_sense_back_left]

See Also `sim_rls`

References Khepera User Manual, Version 4.06

sim_skp

Purpose Set a position for a simulated Khepera to reach

Synopsis `sim_skp(port_id,pos_left,pos_right)`

Description `sim_skp` indicates to the wheel position controller an absolute position to be reached by a simulated Khepera. `port_id` is not used in the function but is preserved to maintain compatibility with `skp`. The units of `pos_left` and `pos_right` are the pulse, each of which is 0.08mm of physical distance. This equates to 125 pulses per millimetre. The maximum number, in pulses, that can be given is 2^{32}-2 which translates to 670 metres distance. Movement occurs in the simulator immediately after receipt of this command every time `kheprom` it called. In the event that another command is being executed when this command is sent, newer commands always supersede older commands.

See Also `sim_spc,sim_rpc`

References Khepera User Manual, Version 4.06

sim_sms

Purpose Set the speed of the two motors on the Khepera

Synopsis `sim_sms(port_id,speed_left,speed_right)`

Description `sim_sms` sets the speed of the two motors on a simulated Khepera. `port_id` is not used in the function but is preserved to maintain compatibility with `sms`. The simulated Khepera operates with one speed regardless of the values placed in speed_left and speed_right. Additionally, in this release of the simulator (V1.0), the magnitude of the values fed to speed_left and speed_right must be equal. This will result in motion that is linear (forward if both parameters positive, backward if both negative) or rotational (left or right depending on which one of the parameters is negative).

See Also `sim_rms`

References Khepera User Manual, Version 4.06

sim_spc

Purpose Set the position counters of a simulated Khepera

Synopsis `sim_spc(port_id,pos_left,pos_right)`

Description `sim_spc` sets the position counter of each motor of a simulated Khepera. `port_id` is not used in the function but is preserved to maintain compatibility with `spc`. The units of `pos_left` and `pos_right` are the pulse, each of which is 0.08mm of physical distance. This equates to 125 pulses per millimetre.

See Also `sim_skp,sim_rpc`

References Khepera User Manual, Version 4.06

simsense

Purpose

A graphical user interface for measuring and displaying sensor values on a simulated khepera robot.

Synopsis

`simsense`

Description

`simsense` provides an interactive graphical user interface (gui) for the collection and display of data gathered from the proximity and light sensors. Data can be gathered as the simulated khepera moves along a linear trajectory (from 5 cm to 60 cm) or through a rotation (15 degrees to 360 degrees).

Notes:

- The `simsense` gui allows data to be stored when the user clicks on the **Save Data** button. This data is stored using the Matlab `save` command. Specifically, the variables `dv` and `dlgstr` are stored in MAT file format under the filename of the user's choice. `dv` is a matrix of test values taken during a test run on the simulated khepera. Fields in `dv` are arranged as follows:

  Columns 1-2: Khep position counter values at each measurement.
  Columns 3-10: Measured values of light or proximity (depending on test performed), one column per sensor.

  The variable dlgstr contains any textual information entered in the **File Description** box of the gui at the time of saving.

- To read back the data, use the Matlab command `load filename`, where *filename* is the name of the Save file you gave above. If you type `who` at the command prompt, the variables `dv` and `dlgstr` will appear. You can read the contents of `dlgstr` simply by typing `dlgstr` at the Matlab command prompt.

See Also

`tstsense`

References

Khepera User Manual, Version 4.06

skp

Purpose Set a position for the Khepera to reach

Synopsis `skp(port_id,pos_left,pos_right)`

Description `skp` indicates to the wheel position controller an absolute position to be reached through the serial port indicated in `port_id`. Valid ports are 1-4. The units of `pos_left` and `pos_right` are the pulse, each of which is 0.08mm of physical distance. This equates to 125 pulses per millimetre. The maximum number, in pulses, that can be given is 2^{32}-2 which translates to 670 metres distance. Movement occurs immediately on receipt of this command. In the event that another command is being executed when this command is sent, newer commands always supersede older commands subject to acceleration and maximal speed constraints.

See Also `spc,rpc,cpid`

References Khepera User Manual, Version 4.06

sms

Purpose Set the speed of the two motors on the Khepera

Synopsis `sms(port_id,speed_left,speed_right)`

Description `sms` sets the speed of the two motors on the Khepera through the serial port indicated in `port_id`. Valid ports are 1-4. The units of `speed_left` and `speed_right` are the pulse/10ms which translates to 8 millimetres per second. The maximum speed is 127 pulses/10ms or 1 metre/second.

See Also `cspc, rms`

References Khepera User Manual, Version 4.06

spc

Purpose Set the position counters of the Khepera

Synopsis `spc(port_id,pos_left,pos_right)`

Description `spc` sets the position counter of each motor thought the serial port indicated in `port_id`. Valid values for port_id are 1-4. The units of `pos_left` and `pos_right` are the pulse, each of which is 0.08mm of physical distance. This equates to 125 pulses per millimetre.

See Also `skp,rpc,cpid`

References Khepera User Manual, Version 4.06

tstsense

Purpose

A graphical user interface for measuring and displaying sensor values on a real khepera robot.

Synopsis

`tstsense`

Description

`tstsense` provides an interactive graphical user interface (gui) for the collection and display of data gathered from the proximity and light sensors. Data can be gathered as the khepera moves along a linear trajectory (from 5 cm to 60 cm) or through a rotation (15 degrees to 360 degrees).

Notes:

- The `tstsense` gui allows data to be stored when the user clicks on the **Save Data** button. This data is stored using the Matlab `save` command. Specifically, the variables `dv` and `dlgstr` are stored in MAT file format under the filename of the user's choice. `dv` is a matrix of test values taken during a test run on the real khepera. Fields in `dv` are arranged as follows:

  Columns 1-2: Khep position counter values at each measurement.
  Columns 3-10: Measured values of light or proximity (depending on test performed), one column per sensor.

  The variable `dlgstr` contains any textual information entered in the **File Description** box of the gui at the time of saving.

- To read back the data, use the Matlab command `load filename`, where *filename* is the name of the Save file you gave above. If you type `who` at the command prompt, the variables `dv` and `dlgstr` will appear. You can read the contents of `dlgstr` simply by typing `dlgstr` at the Matlab command prompt.

See Also

`simsense`

References

Khepera User Manual, Version 4.06

ulsv

Purpose Update light sensor values on a simulated khepera

Synopsis `ulsv`

Description `ulsv` updates the light sensors on a simulated khepera.

Notes:

- This function does not take `btmap1` as an argument because lamps are placed from the **Roborunner** window. As a result, it is possible to maintain a database of lamp object locations within the **Virtual Playpen** window (using the Matlab command `findobj`) without having to access `btmap1`.

- The calibration of the simulated response from this function is not as straightforward as that of `upsv`. In this case, the light intensity (wattage) becomes a variable. The response is computed from measurements using a 7 watt, tungsten filament light bulb. A range of measurements for differing wattages will be performed in the future.

- The value of the light reading is computed as follows (for each lamp):

  ! First, the value of the light sensor is computed using the straight line path between the sensor and the lamp. This value is based on the magnitude of the distance and the wattage of the lamp. In the simulation, the number is based on empirical measurements.

  ! The angle between each sensor and the lamp is then computed. It is compared to a maximum angle which corresponds to a real sensor's angular light response. If it is found to be less than this maximum angle, it is proportionalized to 90 degrees by the operation *angle=sensor_light_angle! 90/max_angle*. This produces a number between 0 and 90 which is then fed to the cosine function. The result is multiplied by the original straight-line computation of light intensity and the result is a light

intensity reading which compensates for the angular deviation between lamp and sensor.

! In empirical measurements, it was determined that a lamp within a certain angle between itself and a sensor does not have its light intensity depreciated. This was incorporated by introducing the variable `min_angle`.

See Also

`upsv`

References

Khepera User Manual
Version 4.06, Khepera Simulator version 2.0 for Unix systems by Olivier Michel

upsv

Purpose Update proximity sensor values on a simulated khepera

Synopsis `upsv(btmap1)`

Description `upsv` updates the proximity sensors on a simulated khepera. This function receives `btmap1`, a 500X600 matrix whose elements represent the colourmap index of each pixel in the simulated environment. The elements of btmap1 are used to determine the existence and relative proximity of objects in the simulation.

Notes:

- To determine what proximity values were appropriate to assign, several measurements were made (using `tstsense`) of a real khepera approaching a wall (made out of randomly coloured Lego blocks). The sensor graphs for all of the measurements were then combined into an average graph. The values in this graph were used to compose the vector `proxval` in `upsv`. Each entry corresponds to the magnitude (as seen on the averaged graph) of the proximity sensor reading. The distance from the sensor to the object is the vector index value in millimeters.

- The determination of whether an object is present or not and of the appropriate proximity value to assign is done as follows:

  ! The khepera's absolute angle and position are determined from the khepera patch object's *userdata* property (see `kheprom`).

  ! For each sensor, the X and Y values of three lines are computed. Each line originates at the center position of the sensor and scans out at an angle of -7, 0, and 7 degrees with respect to the global angle of the sensor (where the global angle is the sum of the absolute angle of the khepera plus the angle of each sensor with respect to the khepera).

  ! The number of points in each scan line is equal to the length of the `proxval` vector. The X and Y indices of each point in each scan line are used to read a value from

`btmap1` (some conditioning is done on the X and Y values to remove illegal matrix indices). If the value is 44, then an object exists at that point and the value of the `proxval` vector for that point on the line is stored in the particular proximity sensor entry. The algorithm is written such that only the largest entry from `proxval` is kept (corresponding to the nearest object to that particular sensor).

! Ten percent noise is computed for each sensor value reading.

- The number of scan lines can be increased or decreased by modifying range of `delta_theta` in `upsv.m`.

See Also `kheprom,tstsense`

References Khepera User Manual, Version 4.06