3
Sensor Modelling and Calibration

3.1 Introduction

For the Khepera simulator, it is not easy to model the light or proximity responses in a uniform and consistent fashion. This is due to the fact that the sensor values depend to a great extent on the environment. For example, the light response of each sensor depends strongly on the distance between the sensor and the light source, the Wattage of the light source, its colour, vertical position, ambient light, etc. The proximity sensors depend largely on the type of material - its reflectivity is a function of its composition. It also depends on the levels of ambient light (particularly infra-red) that are present during the sampling.

Graphs are supplied by the Khepera manufacturer which provide typical light and proximity measurements for various materials, light sources, etc. However, in order to be useful, a model of these responses should not attempt to reproduce just those measurements. If this were done, research using the model would be limited to the environment in which these measurements were taken and this is highly unreasonable. The opposite extreme is to model the responses in a general way. This was done in [Micheal 96] and it provides a reliable, simplified, yet inflexible generalization of the sensor responses. The middle ground is to model some typical data but leave the model parameters open for adjustment. In this manner, the sensor responses from a real Khepera (placed in the intended environment) can be measured and the model parameters can be adjusted to calibrate the simulation model to the real world.

This section provides examples of calibrating the sensor models to the real world. This section starts off with a detailed analysis of the algorithms contained in the functions `ulsv.m` and `upsv.m`. These are Matlab functions used to model proximity and light sensor responses (respectively) of the Khepera robot. This section assumes some knowledge of Matlab programming - graphics handles and object properties in particular, but is not necessary for understanding. It may be helpful to have a copy of these functions handy for quick reference as this section is read.

3.2 Algorithm Analysis: `ulsv.m`

3.2.1 Model Variables

The following variables and constants were created and incorporated into the model of the light sensor response after analysis of graphs from several experimental runs with a real robot.

`squash, xlate` These two variables are used to determine the shape and position of the curve that is used to model the linear response of the ambient light sensors. The use of these variables is described in detail later.

`maxangle` This variable defines the angular limit at which a light source will have an effect on any individual sensor. Note that the effective light "cone" subtends an angle which is twice this variable.

`amb_lite` This variable represents the ambient light in the experimental environment. This can be determined by reading the sensors without a light source and averaging them.

`cutoff_ang` This variable was introduced into the model after observing that graphs from measured light responses displayed a certain thresholding angle. When the angle between the front-and-centre of the light sensor and the light source was within the "cone" subtended by twice this angle, the light response would drop immediately to saturation level (a measurement of about 50) . As discussed later, the sigmoid function used to emulate the light response performs well enough in most instances to zeroize the this variable.

`noise_mux` This variable represents the percentage of random noise that should be added to the light sensor response. The percentage is calculated on the amplitudes of the light levels computed.

3.2.2 Computation of Light Sensor Response

3.2.2.1 Algorithm Initialization and Setup

The m-file begins with the definition of two constants, `sensor_pos` and `sensor_dir`. The first defines the location of each sensor (in millimetres) with respect to the centre of the robot and assuming the robot is facing right (that is, zero degrees of rotation) and that the centre of the robot is at the origin. We shall call the robot's overall angle with respect to its environment the *absolute angle*. We shall call the robot's overall position with respect to its environment the

absolute position. An absolute angle of zero degrees means the robot is facing right. The second constant defines the direction in which each sensor points when the absolute angle of the robot is zero degrees.

Next, the absolute position and angle (`abs_pos, theta`) of the robot are determined by reading the appropriate sections of the vector stored in the Khepera patch object's userdata property. See `kheprom` in the *Khepera Toolbox User's Manual* for a description of these fields.

An array called `sensor_array` is then created by putting `sensor_pos` through a coordinate transformation using `theta` and `abs_pos`. The array is three columns by eight rows. The first two columns are absolute X and Y positions of each sensor and the third column is the absolute angular direction in which each sensor now points (obtained by adding theta to `sensor_dir`).

Finally, before computation begins, the vector containing the old light values in the userdata property is overwritten by a vector containing ambient light values. If no further changes are made in the algorithm, these values will be written back to the Khepera patch object.

3.2.2.2 Computation of Light Levels

The computation of the light levels is done for each lamp present. If there are no lamps, then no computation is done and the ambient light levels are written back to the simulated Khepera. If there are lamps present, the computation loop is then iterated for each lamp that is on. This is determined using the third bit of each lamp object's userdata property (1=on 0=off).

For each lamp that is present *and* illuminated, the following computations are performed. First, the absolute position of the lamp is obtained from the first two bits of the lamp patch object's userdata property. Next, the Euclidean distance of the lamp from each sensor is computed. A vector called `angle_check` is then computed using the function `cvrtang.m`. This vector contains the angular deviation of the current lamp from the centre-line of each sensor. Now, for each sensor, if the angular deviation of the lamp is less than the maximum angle allowed (that is the lamp is inside of its cone of visibility and `angle_check` < `maxangle`), then the lamp can be seen by that sensor.

At this point, there are two factors affecting the light level computed for each sensor: 1) how far it is from the lamp; and 2) the lamp's angular deviation from the centre-line of the sensor. Figure 1 shows manufacturer's measurements of the light sensor response as a function of linear distance from the sensor (ie directly in front of the sensor, moving back along the centre-line) for a 1 Watt lamp in an otherwise unspecified environment. In this simulator, this curve is modelled by a sigmoid function as follows:

Error!

The shape and position of this function are made accessible for control through the use of the variables `squash` and `xlate`. The first variable, `squash`, is used to shape the function. The default value of 0.03 provides a shape that roughly approximates the 1 Watt response. Smaller values (in very small increments) squash the function down and provide a larger slope. The other variable, `xlate`, positions the centre point of the sigmoid over some desired point on the x-axis. The default value of 175 places the centre of the sigmoid in roughly the same position as that shown for the 1 Watt response. The variable that the responses are stored in, `straight_on`, reflects the fact that this is the light response of a bulb as if the bulb were along the centre-line emanating from the front of the sensor.

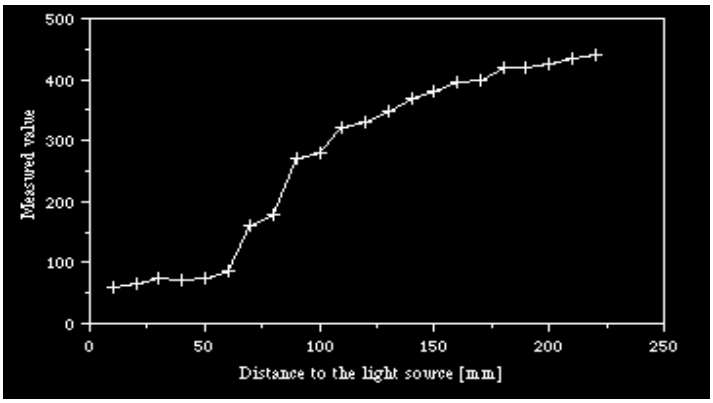

Figure 1
Manufacturer Measurements of Light Sensor
Response with Variable Light Source Distance[1]

The computation above assumes that the lamp is directly in front of the sensor. What is the response as the lamp moves perpendicular across the front of the sensor? As shown in Figure 2 below, the response of each sensor again resembles that of a sigmoid as the angle between the sensor centre-line and a light source decreases.

[1]Figures 1 and 2 courtesy of ftp://lamiftp.epfl.ch/khepera

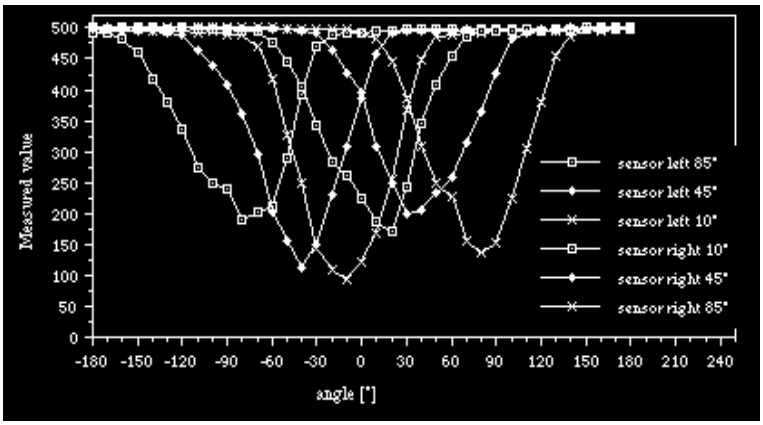

Figure 2
Manufacturer Measurements of Light Sensor
Response with Variable Light Source Angles

Once again, this particular light response is modelled using a sigmoid function and will produce an additive modification factor for the `straight_on` computation done above. When a lamp is not on the centre-line of a sensor but is within the cone defined by `maxangle` and outside the region defined by `cutoff_ang`, the following computation is performed:

$$offcentre = \frac{1}{1+e^{(-4.5*\frac{anglecheck(i)}{maxangle}*2-1)}}$$

The sigmoid is scaled to the domain [0 2] by 4.5 and translation by 1. All of the input angles are scaled on [0 2] by the division of `angle_check` by `maxangle` and then multiplication by 2.

The light response for the straight out distance computed above is combined with the off centre value to compute the total light response of the sensor as shown next:

Error!

There is one other situation in which the value of the light sensor response is computed differently. This occurs when `angle_check < maxangle` (that is, the lamp is within the sensor's visual cone) but `angle_check` is also less than `cutoff_ang`. If `cutoff_ang` is not set to zero, then there will be a region subtended by twice the value of `cutoff_ang` in which the response of the light sensor will always be 50. This additional computation was incorporated to allow the modelling of a sharp "flat" region observed in the light sensor response of a real robot. However, in many instances, the sigmoid function approximation is close enough

to allow this variable to be set to zero (and thereby have no effect).

The responses computed above for each sensor are compared to the values computed for the previous lamp and the lowest values are kept. This corresponds to the brighter lamps overpowering less powerful ones or those that are more distant.

Finally, the computation of each light sensor response is completed by the addition of a percentage of the amplitude as noise. The percentage is specified in `noise_mux`.

3.2.3 Factors Not Modelled

There are two factors affecting light sensor response which have been identified but are not modelled by `ulsv.m`. The first is the effect of reflected light. This was considered too complex to model given the number of configurations of the robot environment and the wide range of reflectivity of the materials used to construct such environments. The second factor is the shadowing or blinding effect of obstacles and walls. This was not modelled because it can be controlled to a certain extent by the vertical placement of lamps in the real environment.

3.3 Algorithm Analysis: `upsv.m`

3.3.1 Computation of Proximity Sensor Response

3.3.1.1 Algorithm Initialization and Setup

The `upsv.m` file begins with the definition of the 20 element vector `proxval`. Each element in this vector corresponds to the magnitude of the proximity sensor response at the distance (in millimetres) corresponding to the element's position in the vector. For example, the ninth element in the vector is 850. This is interpreted by the simulator as a reading of 850 when the sensor is nine millimetres from an object.

The vector `proxval` was determined empirically. The Khepera was made to advance towards a number of different obstacles, both forwards and backwards, and the sensor readings for the front and back two sensors were compiled and averaged. The data was collected using the linear approach tests described in chapter 2.

The length of `proxval` is not fixed. The size of `proxval` is computed after it is defined and is stored in the variable *n*.

The vector `sensedir_data` defines the angular direction in which each sensor is oriented relative to the front of the Khepera. The vector `sense_angles` is defined to indicate the angular position of each sensor on the periphery of the robot from its centre. The definition of each of these vectors is shown in Figure 3.

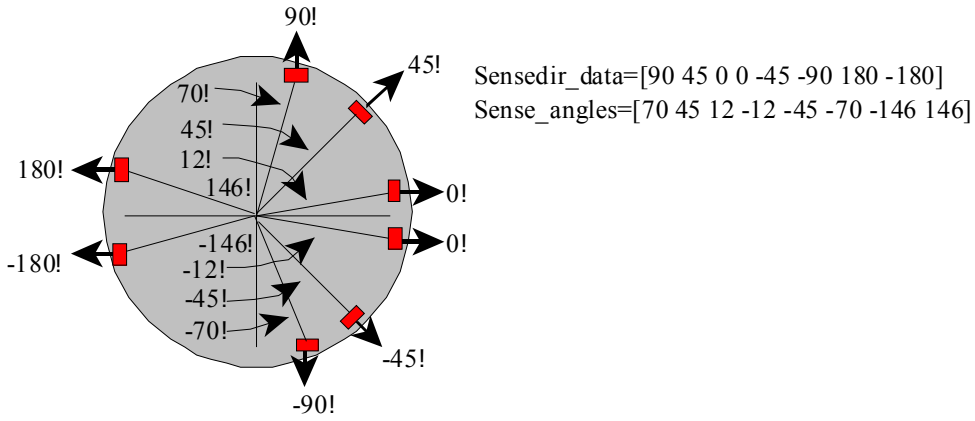

Figure 3
Angular Layout of Khepera Sensors

In final preparation for the computation of the proximity values, the userdata portion of the

Khepera patch object is read to obtain the absolute Khepera position and angle. The initial proximity sensors values are set to zero.

3.3.1.2 Computation of Proximity Levels

Each sensor on the simulated Khepera will scan out *n* millimetres where *n* is equivalent to the size of the vector *proxval*. Each millimetre in the simulation space corresponds to one pixel of the playpen image map. Each sensor will sweep out a cone of 14 degrees directly in front of it. This is done initially by three separate scans at -7, 0 and 7 degrees (with respect to the direction in which the sensor is pointing). To increase the resolution of the search, more scans at lower angular increments could be done.

The computation begins with a **for** loop that iterates from -7 to 7 in increments of 7. Two values, `x_mod` and `y_mod` are computed as the indices into the playpen image map (`btmap1`). These indices are used to determine if any of there is an obstacle present during a scan.

`x_mod` is computed as follows. First, the location of each sensor on the periphery of the Khepera is computed as:

Error!

where θ is the absolute angle of the simulated Khepera. The multiplication factor of 28 is the radius of the Khepera and the operation `ones(1,n)` produces a vector of length *n* to which all elements of the cosine are multiplied. The following computation is then added to this quantity:

`cos((`Θ+*sensedirdata*+*deltatheta*)*π!180)**r*

where `delta_theta` is the **for** loop index which adds -7, 0, or 7 to the sum in the brackets depending on the current iteration of the loop. *r* is a vector of length equal to proxval and each element of *r* is multiplied by the cosine.

Finally, the absolute x-axis position is added to the sum above. A similar operation is done to compute y_mod. Together, x_mod and y_mod form the X and Y axis coordinates for three lines (when all of the loops in the for loop are completed) emanating from each sensor as shown in Figure 4.

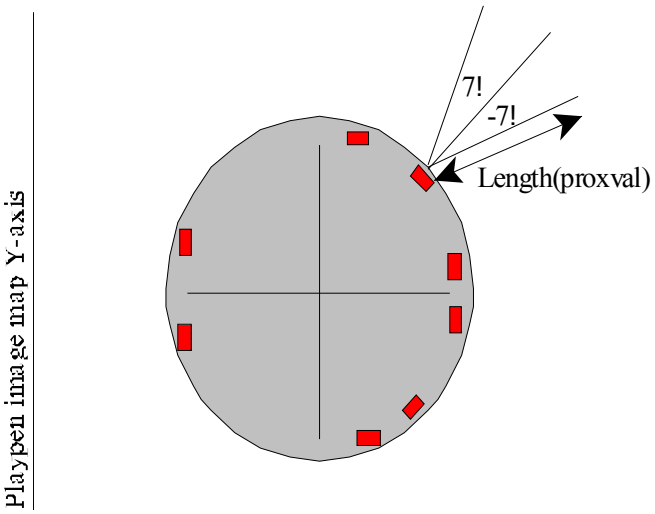

Figure 4
Computation of Proximity Sensor Responses

The X and Y values computed above are checked to ensure that they do not fall outside of the allowable bounds of the of the image map since this would cause an indexing error. For example, if the simulated Khepera were up against the left hand wall in the image map, any of the lines that extended beyond (ie to the left of) the `x=0` position would register as negative image map indices. Since the image map is a matrix, these indices would cause an error.

Using the X and Y coordinates of each of the three lines for each sensor, the `btmap1` image map is indexed. If the value on a particular line is 44 (blue in the image colour map indicating an obstacle), then the position of that value along the line corresponds to the distance of the object from the Khepera and `proxval` is indexed to determine which proximity value to assign. The *nearest* occurance of the value 44 is taken on each line since values further away are irrelevant. Finally, for each of the three lines on each sensor, the largest of the proximity values obtained is kept for storage in the Khepera's proximity sensor registers.

When all of the sensors have had a proximity value computed for it, a small percentage of noise is added to each and then the values are written back to the Khepera patch object *userdata* register (see `kheprom` in the *Reference* chapter).

The granularity of the computation can be increased or decreased by setting the iteration increment in the **for** loop. For example, the default value of 7 will result in three sweep lines. If only one was desired, the loop could be removed. If more lines were required in a cone subtending 20 degrees, the loop could be set up as `for delta_theta=-10:2:10`.

3.3.1.3 Factors Not Modelled

The proximity sensor response computation is very simple. It does not take into account any of the characteristics of active sensing using infrared light sources such as reflectance, interference, or the absorption characteristics of obstacle materials. However, by calibrating the sensor response to the environment through measurement, a good working model can be developed.

3.4 Light Sensor Calibration

3.4.1 Introduction

This section provides, by way of an example, a methodology using the Khepera Toolbox to calibrate the modelled response of the light sensors to that of a real Khepera in its intended environment. By doing this, the simulator can be used to develop a first approximation of any controller intended for use on the real Khepera. This has the advantage that the simulations can be done elsewhere without the requirement to move the physical environment or tying up the robot for extended periods of time (in a many user, single Khepera environment). As well, the simulations can be done faster[2], with more safety, and without periodic supervision. Of course, if only equipped with the Khepera simulator, the default model can be used.

3.4.2 Ambient Light, Linear, and Rotational Tests On Real Khepera

The first step is to gather empirical data from the real Khepera robot. Set up the desired environment for the robot, plug in the robot and start Matlab. From the command prompt, type `rls(pid)` where *pid* is the port number to which the Khepera serial interface is connected (eg. `rls(2)`). A vector of values from each of the light sensors is produced. You may average these values or perhaps use the function amblite.m (function found in util_lib) by typing `amblite(pid)`. This function takes fifty measurements as it rotates. It then reports the average of each sensor and the overall average. This report can be handy for determining if there are under or over-biased sensors. This average will be used to set the ambient light value of the model. Note that this value is highly susceptible to changes in the environment (ie, someone opens a window blind, turns on a lamp, etc) so the environmental conditions should be carefully noted. An example of the amblite output is shown below.

```
>amblite(2)
The mean of each sensor value is:
510   511  510  510  509  509  510  510
The overall mean is: 510
```

The next step is to perform a linear approach test - that is, make the robot approach a light source from the furthest possible position in the experimental environment. This procedure is outlined in the *Khepera Toolbox User's Manual* (note that the simulation environment has a maximum linear approach distance of about 50 cm depending on how the test is set up). For the best possible results, set the **Max Speed** variables in the **Real Khepera Sensor Data Collector** window to 4 in order to slow the robot down. A graph is produced for the right 10 degree sensor as shown in Figure 5.

[2]This, of course, binds this author to the implied contract that the simulator will be optimized for speed in the near future.

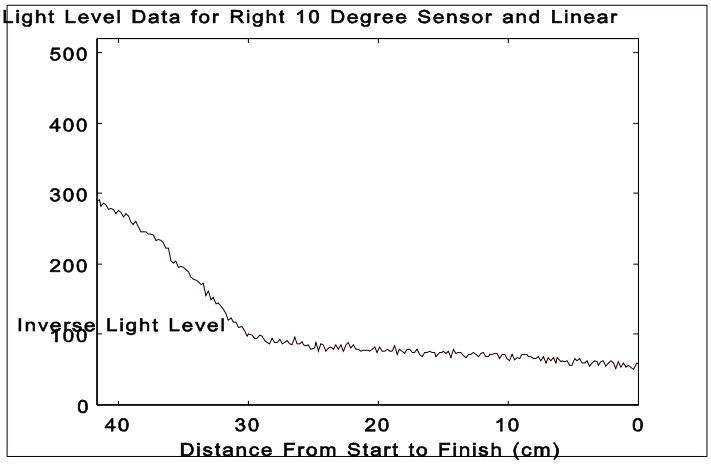

Figure 5
Light Sensor Response For Linear Approach
of Real Khepera to Light Source 40 cm Away

Finally, a rotational test is conducted on the Khepera (as outlined in the *Khepera Toolbox User's Manual*) from a predetermined distance (10 cm in this case) from the light source. Several rotational tests can be conducted, each from a different distance away from the light source, but one should be sufficient. The graph of the response for the back left sensor in our running example is shown below.

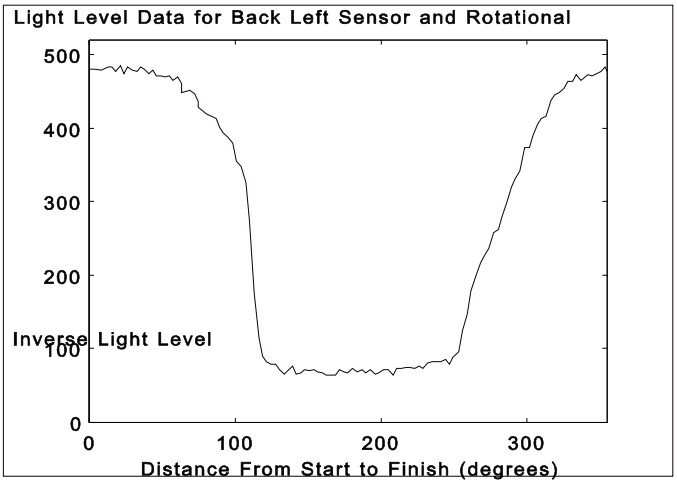

Figure 6
Light Sensor Response For Rotational Test
of Real Khepera at 10 cm From Light Source

3.4.3 Shaping the Simulator Model

The first step in shaping the simulator model is to enter the value of the ambient light determined previously in the **Amb Light** box of the **Sensor Data Collector** window. A linear test must then conducted just as was done for the real robot) by filling in the appropriate control menus of the Simulated Khepera Sensor Data Collector Window as shown below in Figure 7. The default parameters for the other model variables (**Max Dist**, **Max Angle**, **Cut Off**, **Noise**) should be used during this first try.

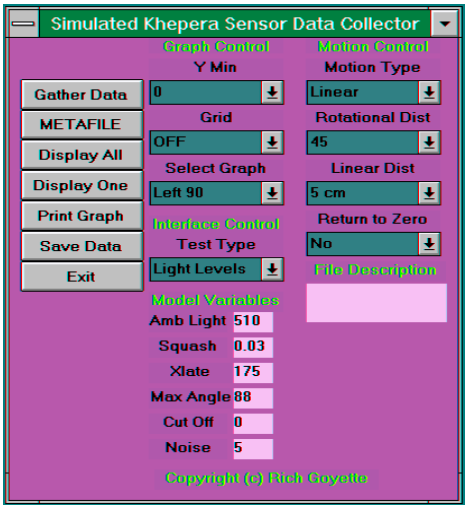

Figure 7
The Simulated Khepera Sensor Data Collector Window

The resulting graph of the right front sensor for the linear approach test on the simulated Khepera is shown below.

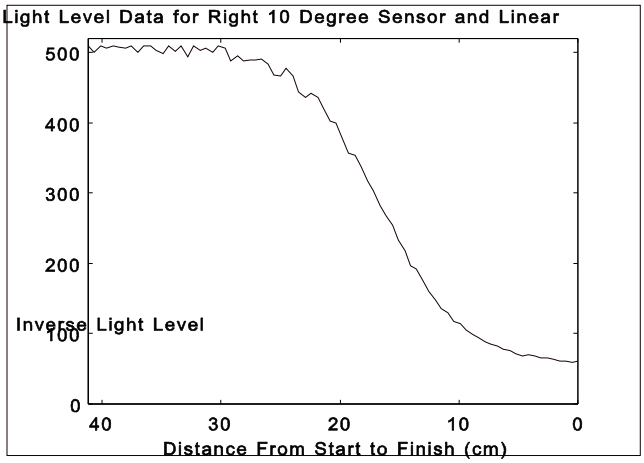

Figure 8
Light Sensor Response For Linear Approach
of Simulated Khepera to Light Source 40 cm Away

The factors affecting this response are primarily **Amb_light**, **Squash**, **Xlate**, and **Noise**. By modifying these over several iterations, the response shown in Figure 9 (which is a close approximation to the real response in Figure x) was produced. The values required to produce this response were: **Squash**=0.02, **Xlate**=400.

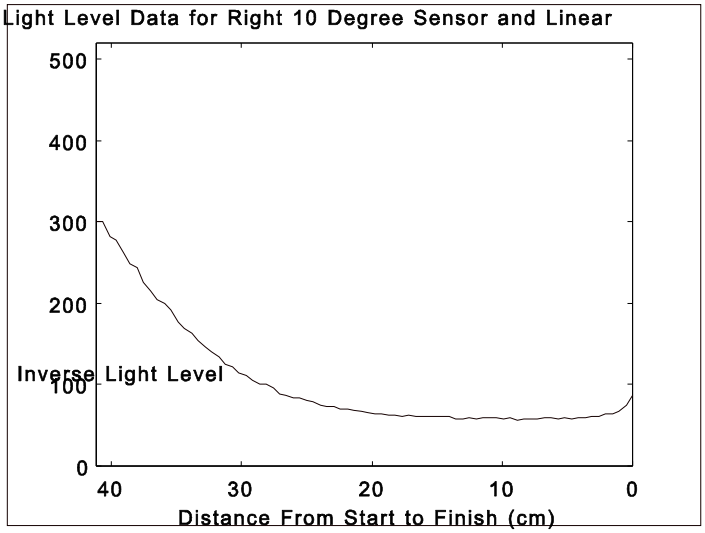

Figure 9
Light Sensor Response For Linear Approach
of Simulated Khepera to Light Source 40 cm Away

The next step is to conduct a rotational test to modify the other variables. Once again, the method of conducting a rotational test is outlined in the *Khepera Toolbox User's Manual*. It is

important to note that the simulated test should take place at an equal distance from the light source as was used in with the real robot. The first cut of the response in shown in Figure 10 below.

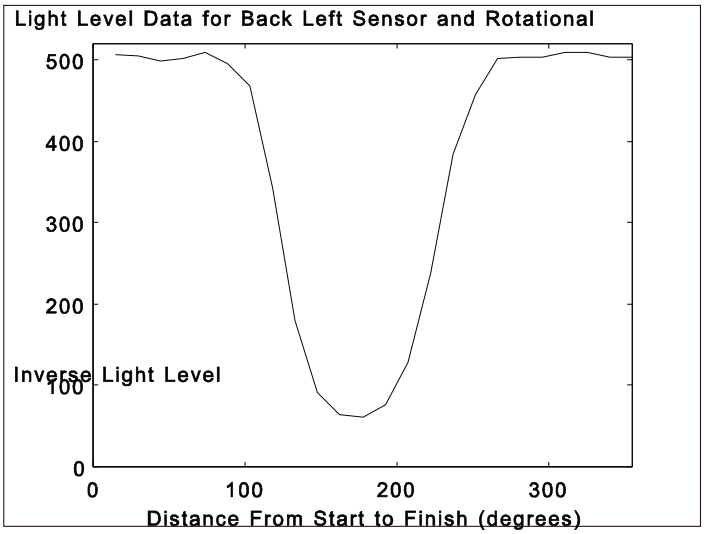

Figure 10
Light Sensor Response For Rotational Test
of Simulated Khepera at 10 cm From Light Source

Comparing Figure 10 with Figure 6, it can be seen that the real sensors appear to have a much wider light cone (effectively about 200 degrees) while the modelled Khepera has a cone of about 170 degrees (**max_angle**=88 degrees, the default). Also note the flat response across the bottom of Figure 6. Figure 11 below shows graphs of the response when **max_angle**=120 and when **cut_off**=0 and **cut_off**=60. Note that the response when **cut_off**=60 is a much closer approximation to Figure 6 than the one where **cut_off**=0. However, because cut_off is a thresholding variable, it introduces jump discontinuities in all sensor responses. This can be undesirable (or, alternatively, can be viewed as a source of noise).

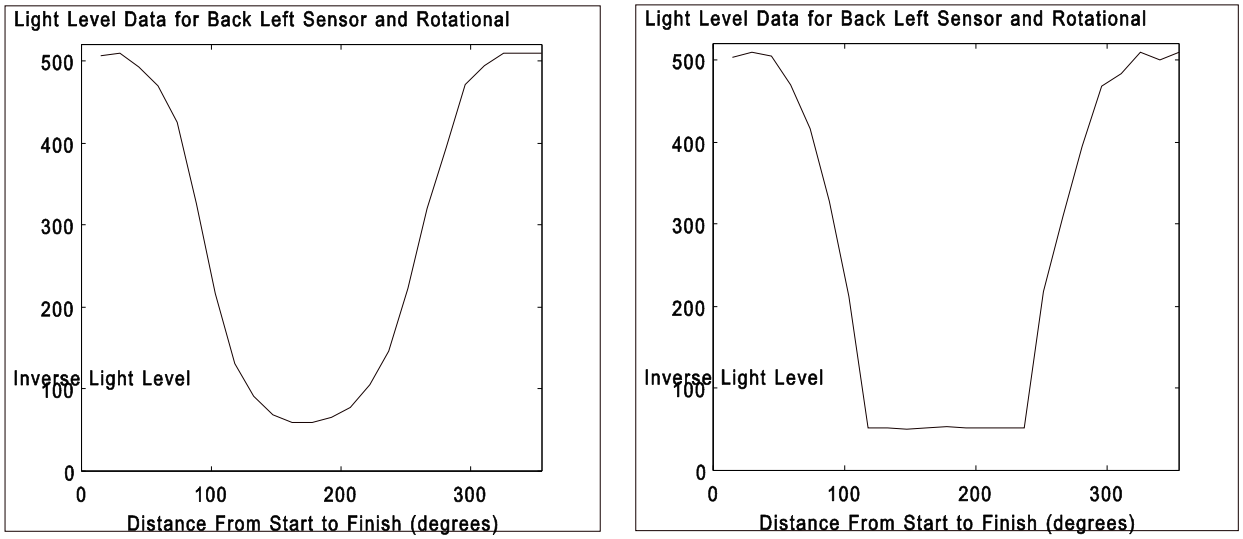

Figure 11
Light Sensor Responses for Rotational Test
of Simulated Khepera at 10 cm From Light Source
With **cut_off**=0 and **cut_off**=60

3.4.4 Comparison of Results

Figure 12 shows two graphs of all of the sensors. The left hand graph is for a linear approach of a real Khepera to the actual light source. The right hand graph shows the modelled response. The front two sensors are modelled quite well. The 45 and 90 degree sensors show promising results at the outset but the 45 degree sensors curve sharply upwards near the end while the 90 degree sensors dip some at the centre and then return to the ambient levels. This is due to the fact that both light sources and Khepera sensors are modelled as point sources (that is, they have no physical dimension in the model). As a result, the response of each sensor as the Khepera gets very close to a lamp does not correlate well with the real world response; the response is computed by adding to the linear distance response a factor which depends on the angle between the light source and the sensor. As a lamp gets very close to a particular sensor (and if it does not lie exactly on the centre-line of the sensor), the angle between the centre of the lamp and the centre of the sensor will begin to increase at an ever faster rate. This will result in a sudden rise in the light level perceived by the sensor because it will look like the lamp is moving out of its vision cone. The solution to this close range blindness issue has been considered but it is desired to keep the modelling functions continuous, smooth, and generally well behaved. The introduction of a threshold to force the desired behaviour at close range (0 to 20 cm) will require further analysis.

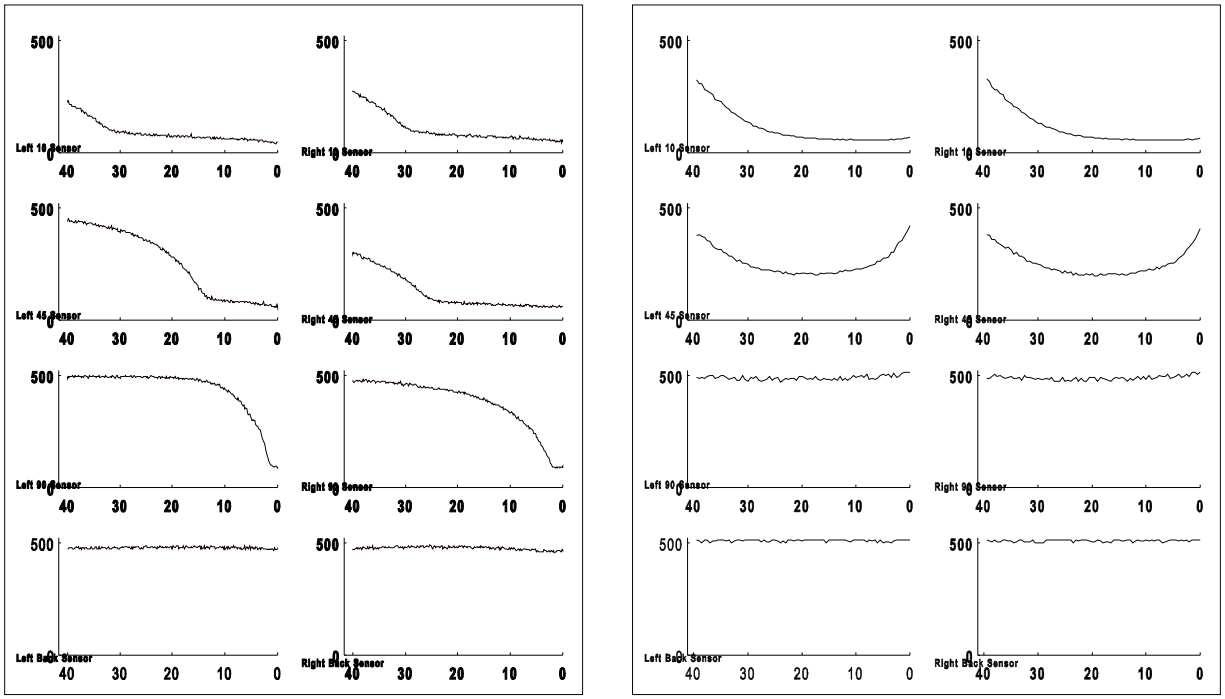

Figure 12
Comparison of Results
for Light Sensor Calibration Example

3.5 Proximity Sensor Calibration

As mentioned previously, the proximity sensor response in the function `upsv.m` was determined empirically by performing a number of linear approach tests and averaging the response of several sensors over these tests. The procedure for doing these tests is quite straight forward and will not be described here.

To incorporate the averaged data into the model, it is a simple matter of opening the m-file `upsv.m` with a text editor and modifying the line that assigns the values to the vector `proxval`.

For example, let us assume that the following data vector was the averaged response of the real Khepera's sensors when approaching some indigenous obstacle of the real environment:

```
sampled_data=[1024 1024 800 300 50]
```

Of course, the data above would indicate that the obstacles of the environment absorb infrared to a large degree since the sensors can only detect the presence of an obstacle 5 millimetres away.

In any event, the m-file entry

```
proxval=[1024 1024 1024 1024 1024 1024 1024 1000 850 700 550 400
250 175 125 50 30 20 10 5]
```

would be replaced by

```
proxval=[1024 1024 800 300 50]
```

There is no need to modify any other parameters because the function upsv.m uses size information determined directly from this vector.