Automatic Sorting Conveyor

Course:	Grundlæggende Indlejrede Systemer (GiS)
	Modul 1, Hardware og Software Grænseflader
Group:	Group 3
Date:	April 09
Names:	Thomas Ginnerup Kristensen
	Henrik Arentoft Dammand
	Anders Hvidgaard Poder

1 Introduction

The Automatic Sorting Conveyer is a computerized sorting system, which sort out defective items during production of transparent blocks. It relieves human operators of the defective item selection during operation. The system consists of three major parts; a motor driving the conveyer belt including an On/Off button, a detection device that detect non-transparent blocks and a deflector arm that weeds out defective items on the conveyer belt. An illustration of this may be seen in Figure 1.

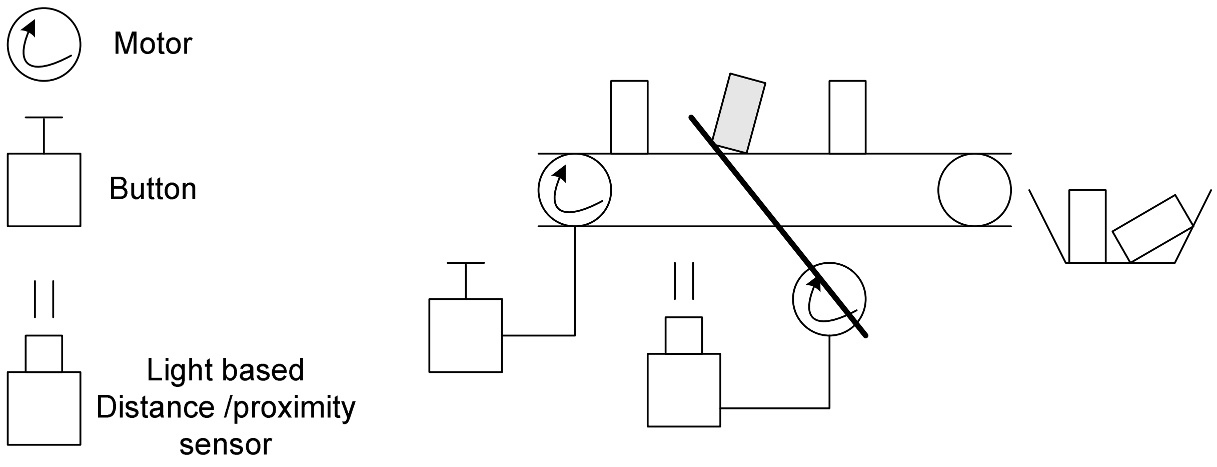


Figure 1: Illustration of Automatic Sorting Conveyor

1.1 Objectives

The objectives of this first report are to investigate and get operational experience with the sensors and actuators from the LEGO NXT. This should give an insight into whether the system is realizable using the LEGO NXT and how it should be implemented.

In short the objectives are to investigate:

- The touch sensor
- The motor used as actuator for the conveyor belt
- The motor used as actuator for the deflector arm
- The light sensor

2 Analysis

This section contains definition of the system and its objects. The system is defined using OOAD including the FACTOR breakdown.

2.1 Problem domain analysis

2.1.1 Problem definition

Refer to the introduction for a description of the problem. However this report focuses on experimentation and not functionality. The reports problem domain (not the project) is therefore merely to gain knowledge. Please refer to 1.1 for a more detailed description of the scope of the report.

2.1.1.1 Class diagram

The sorting system can be modelled into a class diagram composed of a LightSensor class, Button class, Deflector class and Belt class.

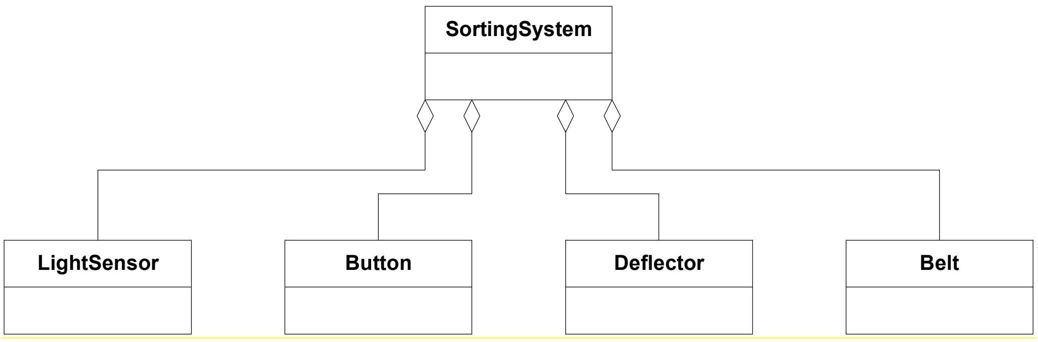


Figure 2: Overview of the classes in Automatic Sorting Conveyor system

2.1.2 FACTOR

2.1.2.1 Functionality for end use

The system is to replace the need for manual detection and removal of defective items. The system shall not be slower than a human operator performing the same task.

2.1.2.2 Application domain of end use

The light sensor shall sample the light level and determine if a non-transparent item is on the conveyer belt. If so the light sensor shall notify the deflector which removes the non-transparent item.

The touch sensor shall start or stop the conveyer belt motor and thereby respectively starting or stopping the system.

2.1.2.3 Conditions for success

According to the objectives for this first project, it is to investigate the sensors and actuators available in the LEGO NXT. The conditions for success are that all project participants have gained operational experience with the NXT sensors and actuators.

2.1.2.4 Technology to be used

The technology will be based on LeJOS and java. The system will be developed in Eclipse environment and executed on LEGO NXT controller.

2.1.2.5 Object system

The main objects in the domain: light sensor, conveyor belt, deflector arm and touch sensor (button).

2.1.2.6 Realization conditions

The system must be implemented using the LEGO NXT controller, sensors and actuators.

2.1.3 Architectural design

2.1.3.1 Class diagram

The class diagram in figure 3 outlines the overall software architecture in Automatic Sorting Conveyor system. The architecture is divided into three layer: Interface, Functionality and Model.

- Interface: This layer act as an interface for each object in system.
- Functionality: This layer is where the functionality and behaviour is implemented.
- Model: This layer is where the state of the system is stored and used by the functional layer.

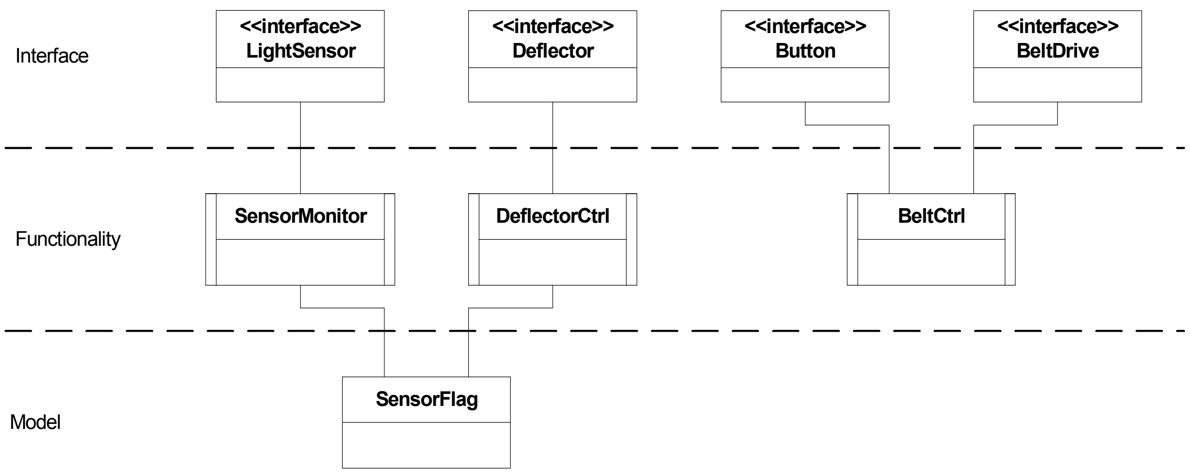


Figure 3: The overall architecture

2.2 Application domain analysis

2.2.1 Use cases

This section has been postponed because this report focuses on experimentation and not functionality.

2.2.2 External Interfaces

The applications External interfaces consists of the sensors and actuators used, i.e.:

- <Interface> LightSensor
- <Interface> Motor (conveyer drive and deflector arm)
- <Interface> TouchSensor

2.2.2.1 Light sensor

The system shall sort out non-transparent blocks and for that task a sensor needs to be able to differentiate between transparent and non-transparent blocks. Since the blocks will be placed on the conveyor belt and will be passing the sensor at some speed, leaving only a short period of time to the detection, the sensors response time must be relatively short. The gap between two consecutive items also directly affects the requirements on the sensors response time.

There are many sensors that may be used to determine the transparency and/or colour of an object, yet within the standard sensor selection of LEGO Mindstorm® the LightSensor is an obvious choice. Choosing a standard component generally has the following advantages and disadvantages.

Advantages

- Readily available.
- Tried and proven.
- Support and API already implemented.
- Cheap in low numbers

Disadvantages

- General purpose, meaning designed for more than just transparency and colour detection. This means that the LightSensor may not be as efficient or accurate as if it was specifically designed for the purpose.
- Expensive in high numbers.

For this assignment the standard LightSensor should be sufficient. In order to know how to best take advantage of the LEGO LightSensor, an analysis of the characteristics of the LightSensor should be performed.

Some of the characteristics of the LightSensor can directly be determined by the LEGO LightSensor specifications, and others are of no importance to the needs of the project. Focus will naturally be on the characteristics required to complete the assignment.

2.2.2.1.1 Characteristics

The LEGO LightSensor can operated in two modes; ambient and reflective. Ambient means that it measures the light level using simple passive measurements. This form of sensor is often used to detect the LUX-count in a room, indicating the room's adequacy for office space with respect to light. The reflective mode is an active mode, where a flood-light (red LED) is activated and the reflected light is measured.

The LightSensor has an ADC converting the measured light, which can then be sampled from software. The output from the ADC is a single integer value, which can be read in one of two ways; raw value or normalized value. The raw value uses configuration to set a high and low reference point, and then outputs values between 0 and 100 relative to these. The normalized value has been processed in some way and also allows for a more accurate reading. The normalized values are between 0 and 1023, with 145 equal dark and 890 equal to sunlight.

Other characteristics are not directly obtainable from the specification, and experiments will have to be conducted to shed some light one the subject.

2.2.2.2 Deflector arm (Motor)

After detection of a non-transparent block the deflector arm must weed out the defective item, leaving all non-detected on the conveyor belt. The required speed of the deflector arm is directly proportional to the speed of the conveyor belt and the minimum gap between two consecutive items. When the actuator rotates the deflector arm swiftly it might not stop immediately, since both motor and deflector arm have inertia. It is necessary that rotation of the deflector arm is roughly accurate i.e. when the deflector arm swings 45 degrees we can trust it has swung 45 degrees within a margin of few degrees, say, ± 2 degrees.

The deflector class models the working of the deflector arm. The class scans the SensorFlag class and takes action according to the status of the SensorFlag class.

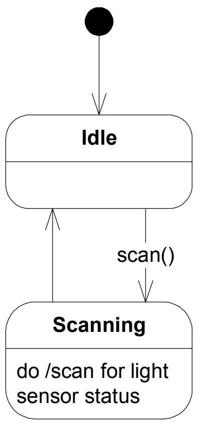


Figure 4: State chart diagram of DeflectorCtrl class

2.2.2.3 Conveyer Belt (Motor)

The conveyer belt is driven by a motor and when the motor is started the system must be fully operational. A simple motor can be used for the belt drive since there are no requirements specifying that the belt must have adjustable speed. An A/C motor would be a simple and inexpensive choice for the application.

However, for the prototype the standard LEGO Servo motor is selected. The Servo motor is a DC motor with a tachometer sensor, which is a flexible and widely used motor in many applications.

The lejOS supports three motors (A, B and C) using the "Motor" class.

The Motor API supports speed control which continuously reads the motors tachometer and adjusts the motor power accordingly.

However a process is spawned to support this speed control and this should be considered when selecting the scheduling strategy.

The Motor class also supports a simple "forward" functionality, which runs the motor at maximum speed with no speed control enabled.

2.2.2.4 Touch sensor

The Touch sensor is a simple switch which can be in one of two states; on or off.

A simple monitoring strategy is to periodically poll or sample the switch's state; on / off.

Two things need to be considered when polling a switch:

- The polling frequency needs to be determined so the system has an acceptable responsiveness.
- A strategy for handling noise/prell on the switch output.

The handling of prell can be implemented in software, by having a relatively high sampling rate and only consider a change in the switch's state, when having detected the same value for e.g. 5-10 samples.

Another approach is to use dedicated HW to handle the prell, e.g. by incorporate a Schmitt-trigger circuit in the switch.

The trade off here is the need for higher sampling rates vs. a higher price for the switch.

Please Note that the LEGO Touch sensor comes with a Schmitt-trigger circuit; hence the software needn't consider prell.

2.2.3 Functions

NA

3 Detailed design

3.1 The Touch sensor

Using the lejOS it's possible to setup an event listener to get the sensors state as a call-back in the application without using polling from the application. However, when using the event listeners in lejOS, a listener thread is spawned and does the actual polling work.

This must go into the scheduling strategy considerations because it will affect the timing of other periodic processes. If it is decided to use Cyclic Executive scheduling in the end application, it should be considered to make the polling from the application in order to make the timings deterministic.

3.2 The conveyer belt motor

In order to simplify the implementation, the motor is operated at maximum speed with no speed regulation. The lejOS API supports this via the "forward()" method.

However the stability of the speed of the conveyer belt is important. This is because the time between the detection of a defective item to the deflector arm is activated is highly coupled to the time it takes the item to travel from the sensor to the deflector arm, hence the belt speed.

When operating the NXT motors at maximum speed, the actual speed is proportional to the battery power. With this in mind the NXT controller must be operated fully charged and with the charger connected.

If a speed control were to be added, e.g. if the deflector arm or the sensor do not react fast enough, it should be considered to handle the speed control from the application. This is because the timing of other tasks will be influenced by the build-in speed control process. If it is decided that the application must control the speed, the power vs. speed ratio should be investigated, using the motors build-in tachometer.

3.3 The deflector arm motor

The LEGO actuator comes with a built in 9V DC motor with a tachometer sensor. LejOS provides three instances, one for each actuator, with methods for controlling speed, angle and direction etc. The deflector arm must be swift; hence the actuator must rotate at highest possible speed (170 rpm @ 9V). If cyclic executive scheduling is chosen it means blocking operation is not desirable and affect what methods should be used. DeflectorCtrl class controls the deflector arm and decides, through SensorFlag, when the deflector should be activated. In the experimentation part of the project the DeflectorCtrl is replaced with DeflectorTest, which is not dependent on external interfaces.

3.4 The light sensor

The LEGO light sensor comes with an API for directly reading the light-level sensed by the light sensor. Sampling the light-sensor and determining if a non-transparent object is located on the conveyor belt is wrapped in the SensorMonitor class, which updates the SensorFlag model accordingly. The light sensor used by the SensorMonitor is dependency injected into the class, thereby allowing for the configuration of the sensor (choosing port, etc.) may be done outside of the class along with configuration of the other sensors/actuators.

As no unit-testing is to be done for this experimental part of the project, no interface wrapping has been created for the external interface LightSensor, thereby allowing the external interface to be stubbed. This should naturally be done if unit-testing was to be performed.

The SensorMonitor has a single run method, which will sample the light sensor, perform needed calculations (running average, dynamic configuration, etc.) and update the SensorFlag accordingly. The actual implementation of the required calculations is not part of this rapport, as it is based on the results of the experiments in this rapport. The single run method is suitable for both the cyclic executive scheduling scheme and a multiple thread scheduling scheme, though not without code modifications (the delay must be controlled by the run method for the multiple thread scheduling scheme, but by the executer in the cyclic executive scheduling scheme).

The SensorMonitor will perform no-object calibration in its constructor, and it is therefore imperative that the external lights and fixed light source is in a stable state at program start-up.

In the experimentation part of the project the SensorMonitor is replaced with LightSensorMeasurements, which is a menu-based test application that does not use the SensorFlag.

4 Test and Experimentation

4.1 The Touch sensor

The analysis consists of a small program which shall determine if the lejOS supported event listener is sufficient to get good responsiveness or if it is necessary to implement a polling scheme from the application. Please refer to the Appendix for further details on the test program.

4.1.1 Result and Conclusion

The program shows that the lejOS support for Touch sensor using event listeners is in fact highly responsive and the conclusion is therefore that the event listener can be used for the application.

4.2 The deflector arm

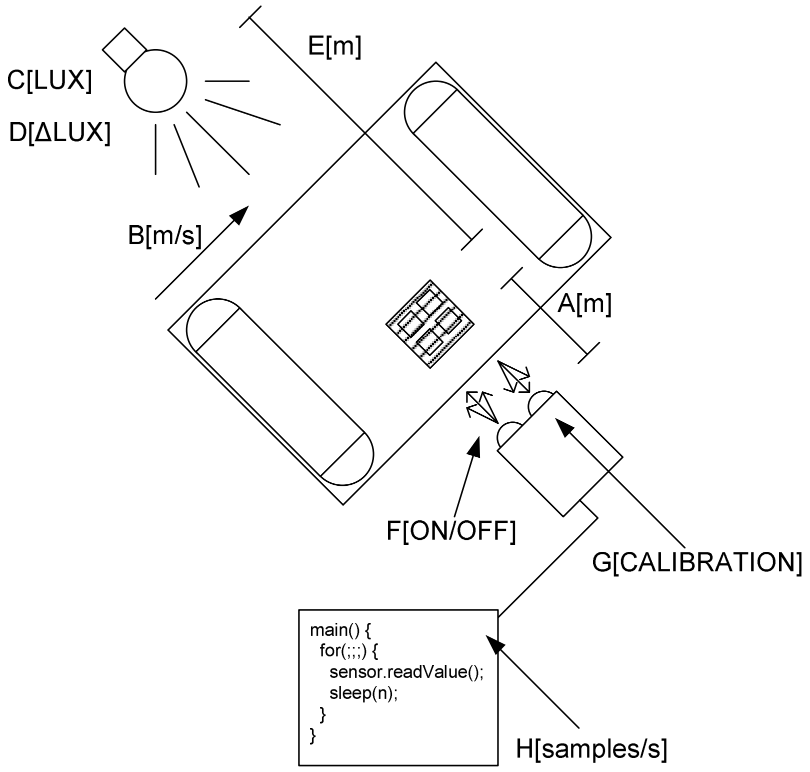
The experiment investigates how much the deflector arm will deviate when it rotates 45 degrees and back again due to inertia in the actuator and the deflector arm. The experiment consists of a small program, called DeflectorTest, that rotates the deflector and displays angle on the display. Please refer to the Appendix for further details on the test program.

4.2.1 Result and conclusion

The analysis has showed following things. First, the actuator is able to rotate the deflector arm 45 degrees and keep deviation within a margin of ± 2 degrees. Second, the methods in lejOS used in the test program are able to regulate the actuator accurately. However, deflector arm might wander over time i.e a small deviation might accumulate which should be considered.

4.3 Light sensor

As may be seen in Figure 5 and Table 1, there are many parameters which may be modified. Some of these we choose to disregard, where others are of no interest. In the following sections we will dive into determining which characteristics to clarify, which to disregard and which to determine by reasoning.



```
main() {
  for(;;;) {
    sensor.readValue();
    sleep(n);
  }
}
```

Figure 5: lightSensor analysis set-up

ID	Description
A	The distance from the light sensor to the object it is sensing. This is important to measure in order to determine the optimal location of the light sensor relative to the objects on the conveyer-belt, so as to achieve the most accurate readings. This may be tested by moving

	the light sensor relative to the object and logging the reading-accuracy.
B	The speed of the conveyer-belt. This is very important, as the sampling frequency and the response-time of the light sensor is directly related to the maximum speed of the conveyer-belt. To determine the response-time, a simple program sampling at maximum speed may be written, and the speed of the conveyer-belt increased until sampling fails (either due to sampling frequency insufficient or response-time issues).
C	The light in the room may have profound effect on the light sensors accuracy, and this may be tested by running identical tests, but with different external light levels. (no sampling should be done during the level change).
D	The change in lighting may also affect the accuracy of the readings. This can be determined by sampling while the lighting is being turned up or down.
E	The location of the external light source relative to the object being sensed and the light sensor is also important. This may be measured by moving the light source while sampling.
F	Whether the flood-light of the light sensor is on may affect the accuracy of the readings.
G	The calibration of the light sensor may affect the readings, and testing for the best configuration can be very important, as well as determining configuration drift during operation and the possibility of re-calibration.
H	The sampling frequency is also important; as it may be that the response-time of the sensor is faster than the max sample-rate, making the sample-rate the important configuration or vice versa. It may be determined what sampling frequency can be achieved, also under the condition of servicing other peripherals.

Table 1: light sensor analysis set-up description

4.3.1.1 Modes

It is possible to use the ambient mode to detect transparency, if a stationary light-source is placed across form the sensor and the items to measure is transported between the light source and the sensor.

An alternative is to use the reflective mode to register how much light is reflected off the item, thereby measuring transparency.

Both solutions will be sensitive to changes in the surrounding light-level, as normal light ("white" light), is a broad spectrum light containing elements of all colours, and the NXT light sensor only allow measuring of light levels without filtering, i.e. there is no way to measure the light level of a specific frequency. Even if the NXT light sensor did support filtering, which it does not, it would not fully solve the problem, as “white” light contains all frequencies, and therefore also the frequency included in the filter.

The reflective measurements has a second disadvantage, as convex surface or a straight surface at an angle would not reflect the light back on the sensor, as shown in Figure 6, and therefore appear transparent. The reflective solution would have a big advantage if filtering was possible, as the flood light is a red LED operating in the infra-red light spectrum. If it was possible to filter the light sensor's incoming light (without gluing a physical filter to the sensor), one would only have to shield the light sensor against one specific spectrum of light. Unfortunately, as mentioned before, the NXT light sensor does not support filtering. For these reasons the solution using reflective measurements are discarded. Reflective measurements are well suited for colour or shade measurements, but not for transparency-measurements.

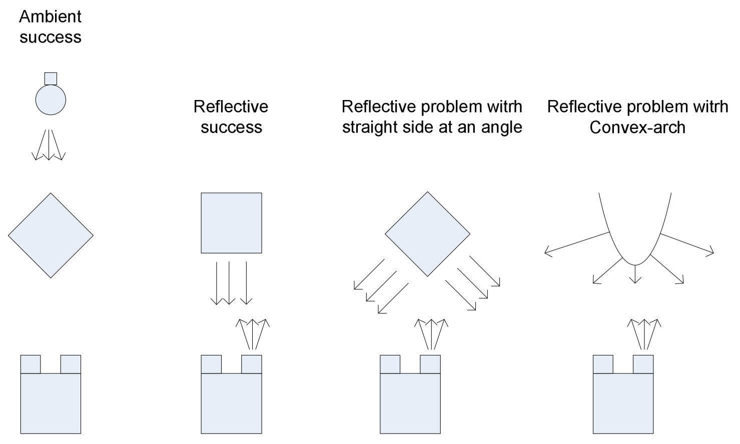

Ambient success
Reflective success
Reflective problem witrh straight side at an angle
Reflective problem witrh Convex-arch

Figure 6: Reflective mode problem

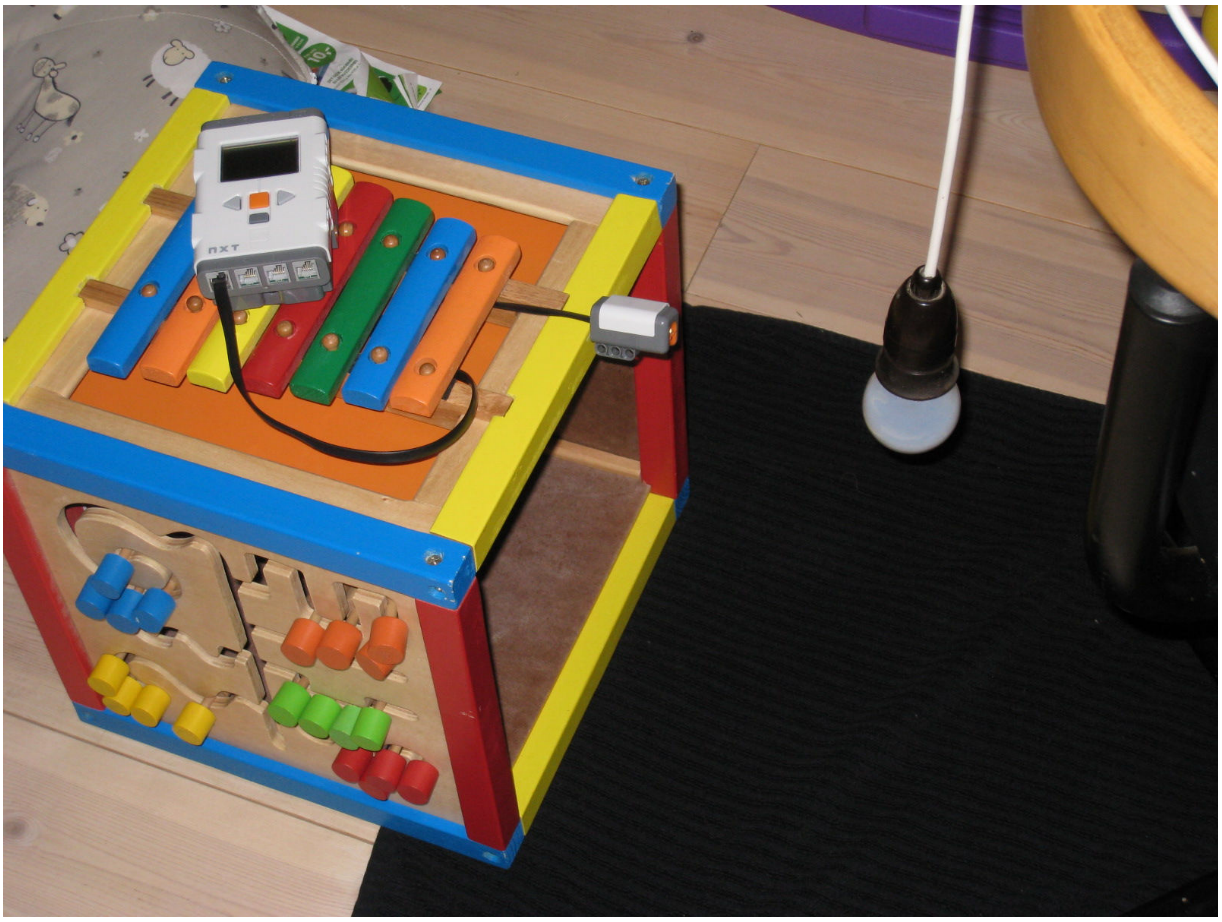
4.3.1.2 Normalised values

The LEGO specification is unclear about what the normalized values actually are, compared to the raw values, and we therefore needs to perform an experiment to determine this.

The reason this is important is that the raw values might be used to achieve higher accuracy through dynamic configuration (even though the LEGO specification dictates that the normalised values give the highest accuracy).

We run a simple experiment to determine what the “configuration” and “normalized value” entails.

The code for the experiment may be seen in Section 6.1.3 where the experiment to run is *Configure.* The set-up is shown in Picture 1.



Picture 1: Normalised test set-up

The test results are shown in Table 2, and the experiment is done with a 40W light-bulb at the specified distances. Configuration is done at the indicated distances (40cm for high, and 50cm for low).

	Configuration high value	Configuration low value	Normalised value	Raw value
40cm	728	NA	728	100
50cm	NA	660	660	0
100cm	NA	NA	527	-195
30cm	NA	NA	779	175

Table 2: Configuration and normalized results

As it may be seen the configuration is merely setting two fixed points on the normalized curve and then returning values relative to that. This is illustrated in Figure 4.

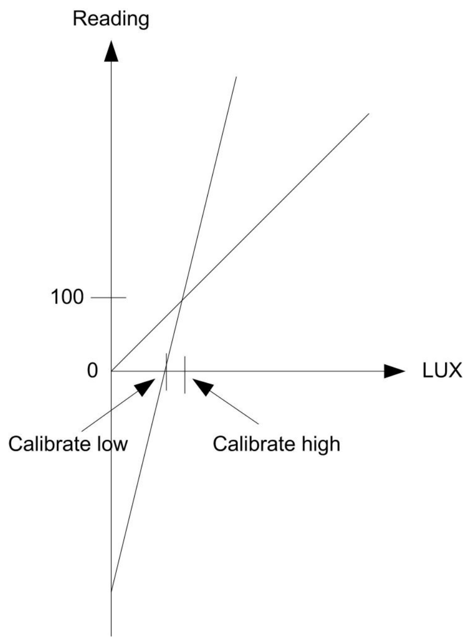


Figure 7: Normalized versus “raw” values

Further experiments where we attempt to get the “raw” value to change without the normalized, by configuring two points close to each other and then altering the light just a little, have been conducted. These have shown that it is not possible to achieve a higher accuracy than the normalized, as the “raw” values only change if the normalized value changes.

This means that there is no reason to use the configurable raw values, as the same accuracy can be achieved with a larger range using normalized values.

For this reason raw values and configuration may be disregarded (at least the form of configuration built in to the LeJOS API).

4.3.1.3 Reaction time

The speed of the internal ADC, as well as the speed of the sample and hold circuit and photosensitive film, is unspecified, yet a simple experiment with a rapidly changing light-source could be use to determine if oversampling is possible, thereby determining if the light sensor response time can become a problem. If oversampling is not possible, the light sensors response time is unimportant, as it is simply “fast enough”.

The code for the experiment may be seen in Section 6.1.3 where the experiment to run is *Reaction time.*

Unfortunately such a fast changing light-source was not possible to achieve, so the experiment has been abandoned and the sensor simply declared “fast enough”.

4.3.1.4 Range

In the LEGO light sensor specification, the description of the range is specified as dark to sunlight. This is a very loose specification, as it does not even detail if they mean direct sunlight (32000-130000LUX) or just sunlight (10000 – 20000LUX). For this reason we are going to attempt to get the lowest and highest value, and also determine where the sensor achieves saturation, if it is at all possible – 130000LUX can be difficult to achieve with simple equipment.

The code for the experiment may be seen in Section 6.1.3 where the experiment to run is *Sampling.* The set-up is shown in Picture 1.

The test results are shown in Table 3, and the experiment is done with a 25/40/60W light-bulb at the specified distances.

	60W	40W	25W	Darkness
0cm	NA	NA	NA	145
5cm	915	916	912	NA
10cm	909	901	902	NA
20cm	902	882	812	NA
30cm	871	779	705	NA
40cm	800	728	654	NA

Table 3: Range measurements

The LEJOS API stipulates that the normalised values are between 145 and 890. We can also see that at 10cm, we get some inconsistent measurements (25W higher then 40W), so it is likely that this is the light sensor's range, meaning that the LEJOS API is correct. For this reason we will determine the range to be between 145 and 890. Further experiments with moving the light-bulbs have shown that the distance at which the maximum range value (890) is achieved is:

- 26cm for 60W
- 21cm for 40W
- 14cm for 25W

It is therefore pointless to perform any measurements closer than the specified distances. The same is most likely true for the lower values, and any values close to 145 should be taken with a grain of salt. Measuring exactly at what value the light sensor starts registering above 145 require equipment beyond the assignment.

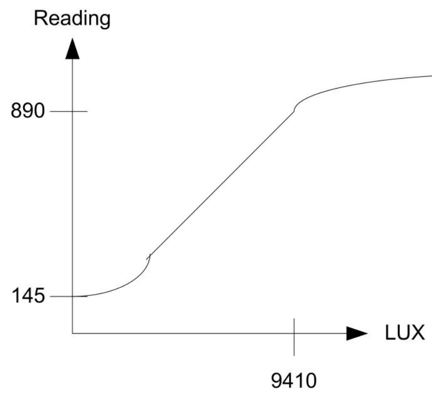


Figure 8: Saturation of light sensor

To validate the LEGO statement of "sunlight" at 890, it is worth mentioning that a 40W light-bulb emit 415LUX at 1m, or approximately $1/(0,21/1)^2 * 415 = 9410$LUX at 21cm, which is just under the 10000LUX, which is the minimum for sunlight.

4.3.1.5 Stability

With no change in the external lighting two consecutive measurements should naturally give the same result, yet the light sensor may have some inaccuracies in its measurements, which means that this is not the case. In order to determine if this is the situation, and how big a variation there is, we run an experiment sampling 10000 times and outputting some key numbers for the samples.

The code for the experiment may be seen in Section 6.1.3 where the experiment to run is *Stability.* The set-up is shown in Picture 1.

The test results are shown in Table 4, and the experiment is done with a 40W light-bulb at 50cm. In Figure 9 may be seen how the samples distribute.

Sampling time for 10000 samples	988ms
Max sample value	696
Min sample value	680
Average value	688

Table 4: Stability measurements

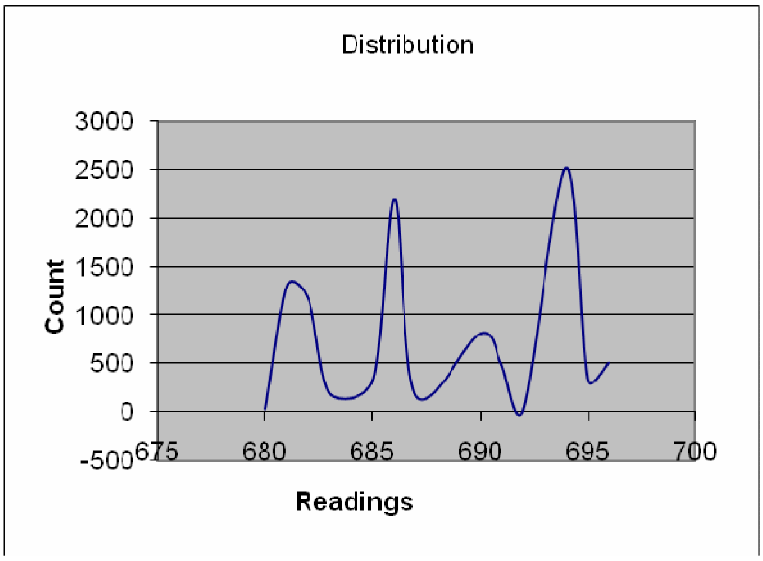


Figure 9: Distribution of the samples

From these results the maximum difference can be determined as 16 (696 – 680). We have to consider this flaw when using our samples, and consider ways of increasing accuracy, e.g. over-sampling (see Section 4.3.1.8). It may be seen that the distribution is not a Normal Distribution, as there is no gausses curve around the average.

Furthermore it should be noted that for this to be universally true the transfer function must be linear, otherwise the maximum difference and distribution may be dependent on the input value.

4.3.1.6 Transfer function

As mentioned above the Normalized mode has a range from 145 to 890, in which it has a certain response to light changes, the question is; what is the transfer function between the actual light level and the read value? Preferably it is linear, but the specifications do not specify this transfer function, so we have to determine it through experimentation.

Determining if an object is located in front to the light sensor does not require linearity, it is more important that it is sensitive to change; determining the point where a change in input causes the highest change in output, see Figure 7

In the Figure, which is merely an example, there is a vast difference in the change in reading for the same change in LUX, which is the very nature of non-linearity.

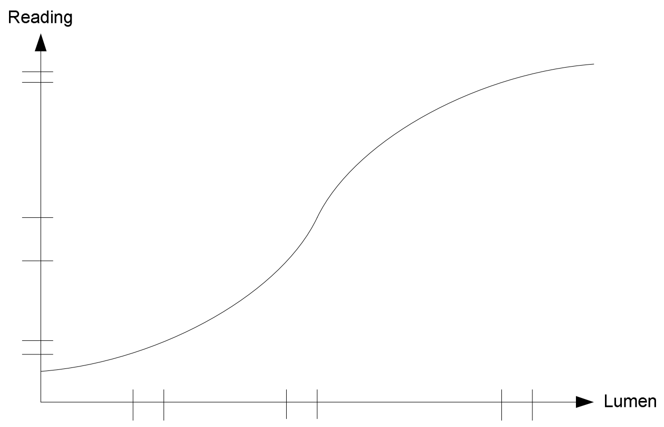


Figure 10: Sensitivity to change (non-linear)

It is also important that this point is in a range where we can control the sensitivity to external changes in light, but we will deal with that in section 4.3.1.7.

In the present requirements we are only interested in determining if there is anything non-transparent on the conveyor belt, yet if we needed to know the level of transparency it becomes important to know the relationship between the measured values and the actual strength of the light; also known as the transfer function.

For these reasons we will experiment with the linearity, and attempt to determine the transfer functions.

In order to shield the experiment from external light sources, and thereby get the best result, it is proposed to perform the measurements in a tube, as shown in Figure 8, set-up 1. Unfortunately this has a disadvantage, as the LUX relative to distance only applies for non-directed light, and a non-reflective tube is difficult to achieve. It may therefore be better to simply have no tube at all, and reply on the fact that if the room is dark and there is a long distance to reflective surfaces, the reflected light will be negligible when it returns to the light sensor, and can therefore be disregarded.

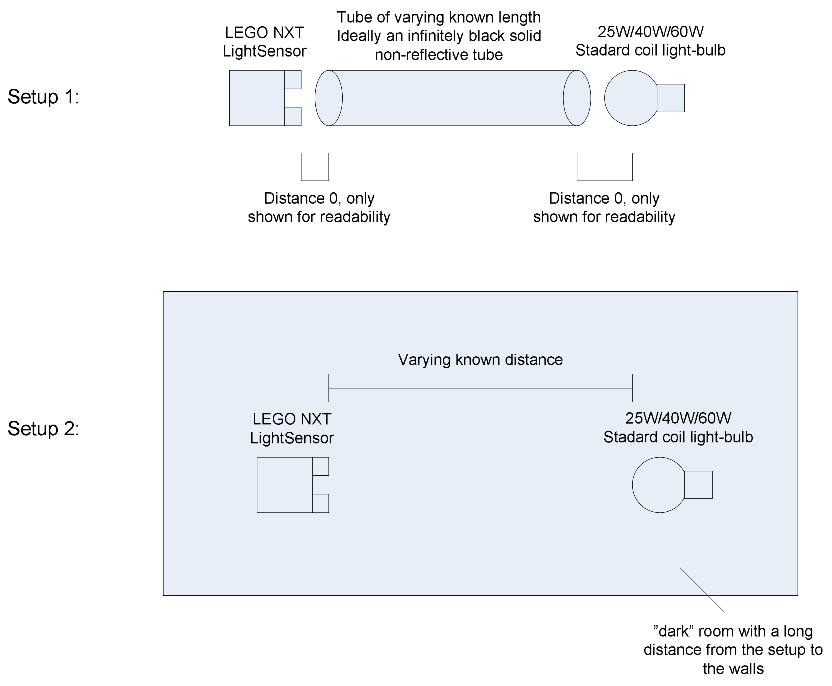
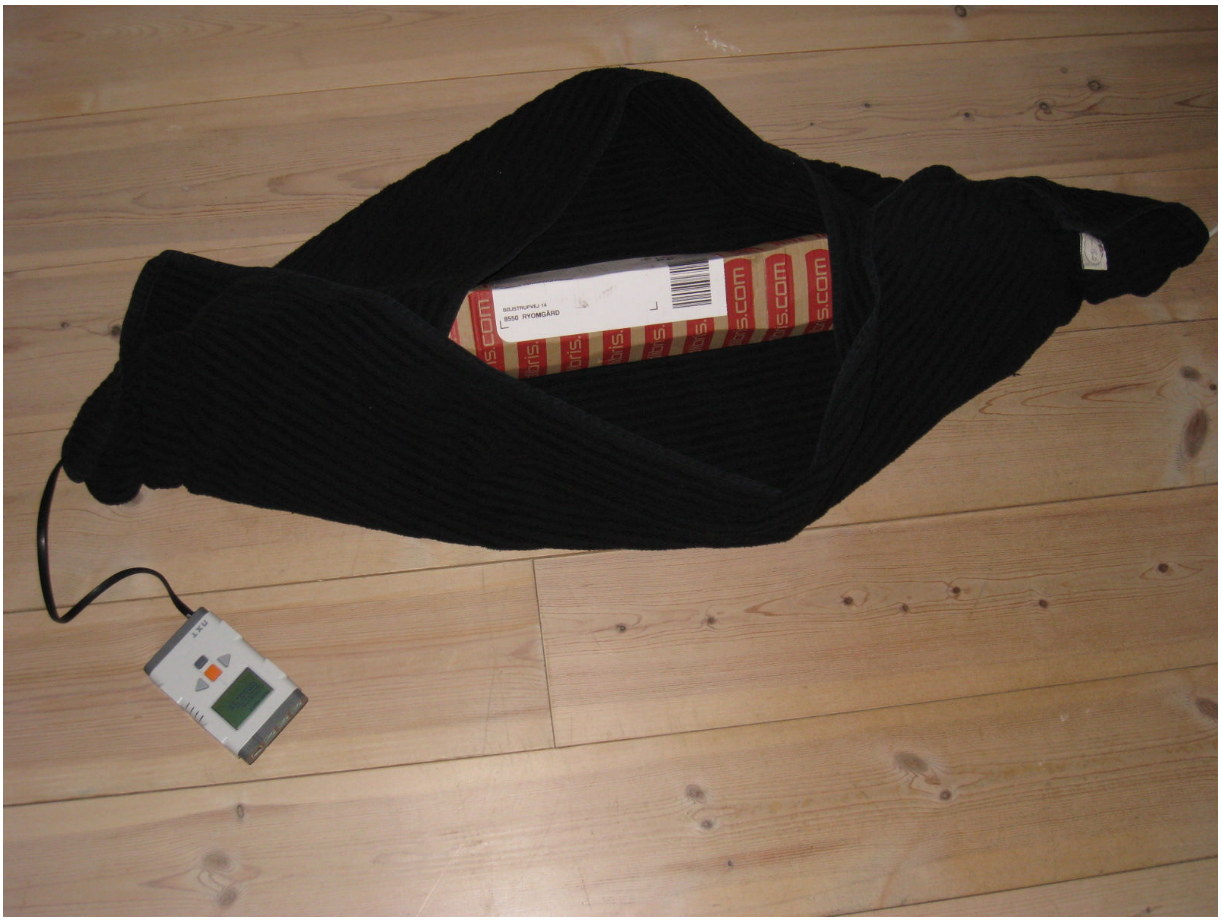

Setup 1:
LEGO NXT LightSensor
Tube of varying known length Ideally an infinitely black solid non-reflective tube
25W/40W/60W Stadard coil light-bulb
Distance 0, only shown for readability
Distance 0, only shown for readability
Setup 2:
Varying known distance
LEGO NXT LightSensor
25W/40W/60W Stadard coil light-bulb
"dark" room with a long distance from the setup to the walls

Figure 11: Transfer function test set-up

In order to determine if a tube is possible we create a tube in the valid range (40W at 40cm), and measure the light reading from the light sensor, see Picture 2. This value should be less than the value without the tube, as it, ideally, removes any external light. The results can be seen in Table 5, and actual test set-up in Picture 2.



Picture 2: Tube test set-up

With tube	900
Without tube	728

Table 5: Tube experiment

From this we may conclude that the tube is in no way non-reflective, and actually far worsen the readings (the light is refracted off the sides of the tube). We will for this reason use test set-up 2, which is the same set-up as has been used before, see Picture 1. The code for the experiment may be seen in Section 6.1.3, where the experiment to run is *Sampling.*

The test results are shown in Table 6.

	60W	40W	25W	Off
40cm	779	749	659	230
70cm	662	629	548	NA

100cm	586	566	493	NA
150cm	526	497	432	NA
200cm	487	462	395	NA

Table 6: Transfer function measurements

As it may be seen the light intensity decrease when the distance increase, which is equivalent to the law that the light intensity decrease relative to the square of the distance. With this information along with the specifications for the light-bulbs used, we can calculate what the LUX values should be at the given distances and light intensity. This is shown in Table 7.

The values at 1m are specified by the manufacturer (see Ref. [2]), and light dissipates relative to the square of the distance. This gives us the equation in Equation 1 for the light intensity at different distances (see Ref. [1]).

$$LUX\_at\_dist\_2 = \frac{1}{\left(\frac{dist\_2}{dist\_1}\right)^2} LUX\_at\_dist\_1$$

Equation 1: LUX at a given distance

	60W	40W	25W
1m	710	415	230
40cm	4438	2594	1438
70cm	1449	847	469
100cm	710	415	230
150cm	316	184	102
200cm	178	104	58

Table 7: Theoretical values

Firstly we have to realize that we have an initial offset of 230, which is the “darkness” that was attainable at the time the test was conducted. We could normalize the readings first (e.g. choose 60W-40cm to be 100, but it would not increase readability, so it has not been done).The results are shown in Figure 12.

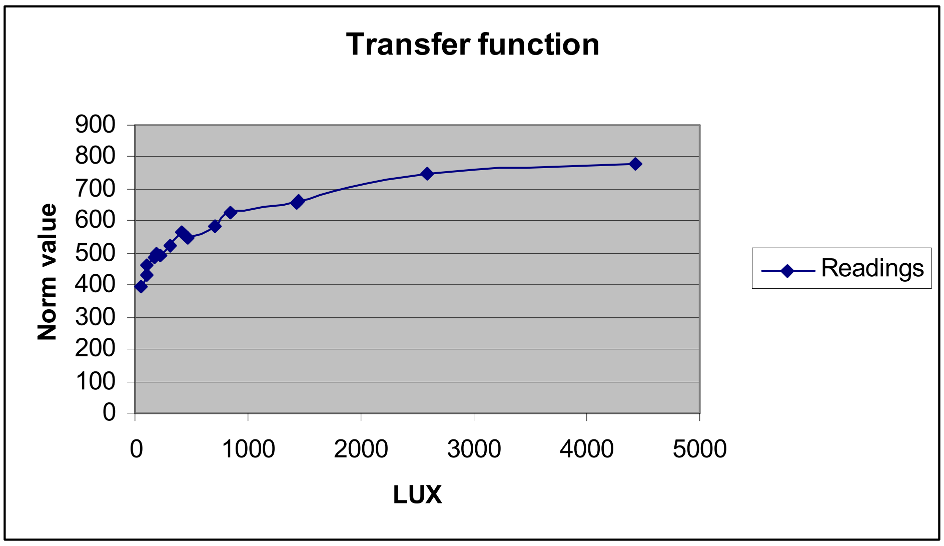


Figure 12: Change in theoretical versus change in actual values

As it may be seen there is definitely not a linear dependency between LUX and the normalized readings from the light sensor. It is, however, possible that "normalizing" means making it linear to distance. We try this theory in Figure 13.

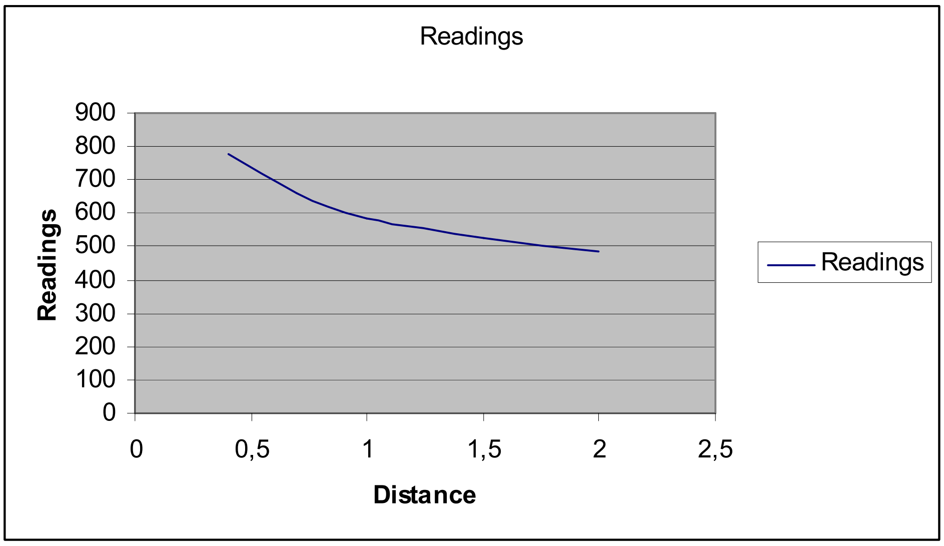


Figure 13: Readings versus distance

Unfortunately it may be seen that this is also not linear, and the "normalization" remains an unknown. Luckily it is not required to know the transfer function, only where the sensitivity to change is largest, which may be seen to be at the lower LUX values (may not be seen from the graph, but can be read from the table)

4.3.1.7 Sensitivity

All the above experiments are done in a relatively dark room, with no changing external lights other then the fixed light, which is only modified as part of the experiment.

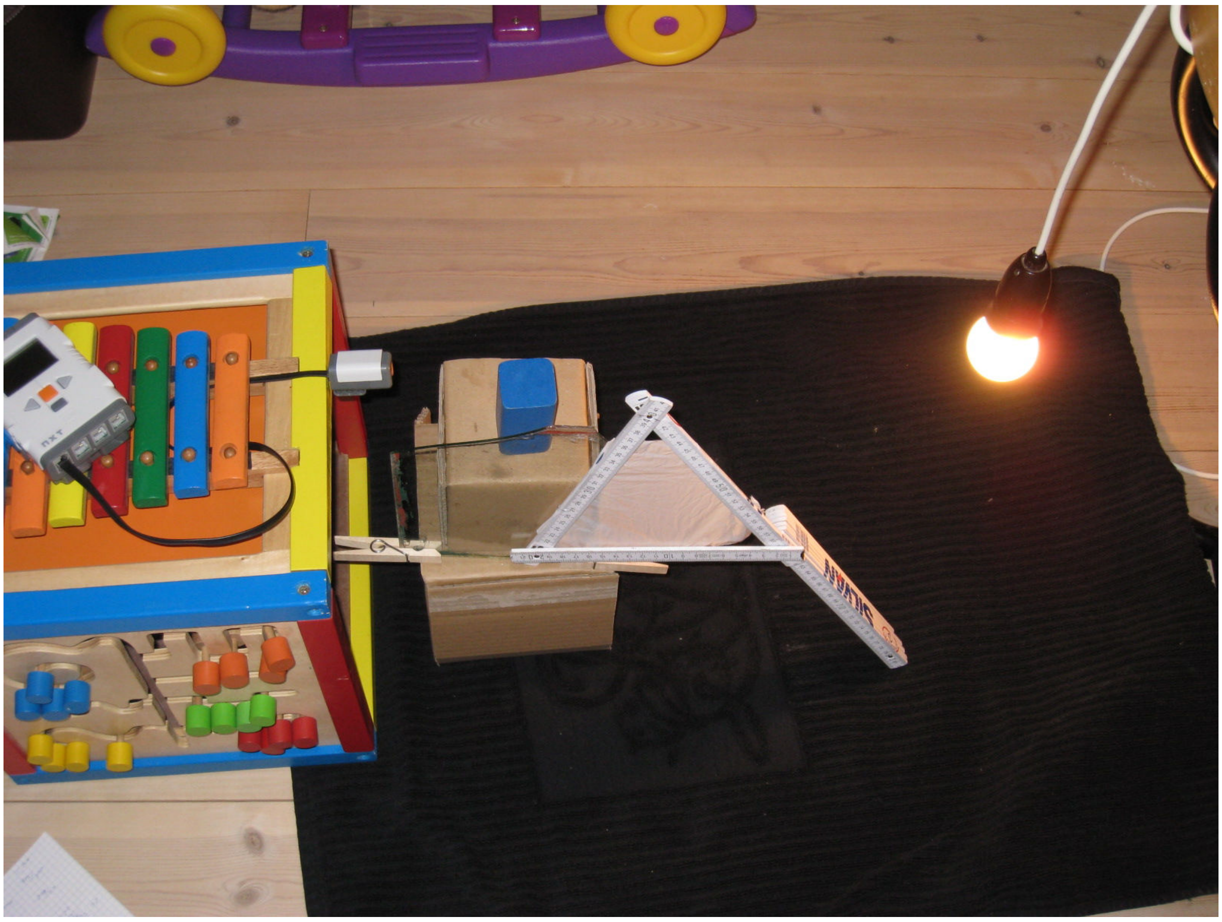
Unfortunately this is not common operating conditions of a sorting conveyor, and the changes in the external lights will affect the light sensor's readings.

Naturally, the stronger the fixed controlled ambient light is, the stronger the changes in the external light have to be to affect it. Based on the results of the transfer function experiment we know that sensitivity is higher at the lower end of the range. However the sensitivity at the higher end of the range is “good enough”, making the sensitivity to external light changes the primary concern, and it therefore makes sense to place the highest possible reading (the no-object reading) at 890, and therefore also do the sensitivity measurements there.

It is obvious that the closer the object is to the light sensor the better for the reading (the more it will shade the light sensor from not only the ambient fixed light but also from external light sources). Naturally it cannot be located directly against the light sensor, as it has to move on a conveyor-belt, yet a distance of 5cm is not unrealistic.

In the test we are using some realistic external light sources, i.e. turning on the light in the room equals weak external light and opening the drapes on a sunny day equals strong external light (the set-up is **not** located in direct sunlight).

We then introduce a non-transparent, semi-transparent and transparent object between the light sensor and the fixed light source (5cm from the light sensor) and measure how it reacts. The code for the experiment may be seen in Section 6.1.3, where the experiment to run is *Sampling.* The set-up is shown in Picture 3.



Picture 3: Sensitivity test set-up

The results may be seen in Table 8.

21cm between sensor and light, 40W external light.	“No” external light	Weak external light	Strong external light
No object	878	886	888
Transparent object (glass)	846	858	864
Semi-transparent object (paper)	811	798	812
Dark object (wooden block)	400	416	451

Table 8: External light sensitivity measurements

As it may be seen the changes in external lighting introduce an extra inaccuracy in the reading (the other may be seen in Section 4.3.1.5). This may be alleviated in a number of ways; shielding the set-up from external light sources, e.g. no windows or filters on the windows, continuous calibration and over-sampling (see section 4.3.1.8), among others.

4.3.1.8 Dynamic calibration and over-sampling

This section does not deal with experiments, but rather with ways of making the light sensor (or rather the program using the light sensor) less sensitive to uncontrolled changes in external light.

Firstly we have to realise that there are several forms of changes that we can deal with.

- Small changes in external light over time.
- Rapid fluxuations in light.
- Large rapid change in external light.

The reason for this breakdown is because the different situations require different solutions.

First, the small changes in external light over time, means that perhaps it is becoming day outside and the external lighting therefore change from dark to indirect sunlight, which is a huge change in LUX, but it happens over a course of several hours. To deal with this we may use the knowledge that the conveyor-belt carries object with nothing in between them. This means that the light sensor has to read its maximum value between objects. If it reads a smaller or larger value it should recalibrate. Determining when the light sensor is “between objects” is not a trivial assignment, and it will be left for later reports, if dynamic calibration is needed.

Second, rapid fluxuations, like from a door opening reflecting sunlight onto the light sensor, or a shadow moving quickly over the sensor, may be handled by over-sampling.

This has been mentioned before under the section about stability (see Section 4.3.1.5), which can also benefit from over-sampling, as we can calculate e.g. a running average to improve stability, and also filter out the rapid fluxuations in external light.

Over-sampling means sampling multiple times within a single object. For this to be possible the sampling and detection of the light sensor must be considerably faster than the movement of the conveyor belt. Assuming the conveyor belt is moving a 1m/s and the object is 5cm long, and we wish to have 10 samples within the object, we must perform sampling at 200Hz, as 5cm is covered in 50ms and 10 samples in 20ms requires one sample per 2ms, or 200Hz. The equation is Hz = (<conveyor speed>[m/s] / <object size>[m]) * <sample count>. Whether this is possible ties to the reaction time of the light sensor (see section 4.3.1.3 (n.b. The over-sampling mentioned in section 4.3.1.3 is not the same as in this section)), for if the sensor is sluggish, meaning that a light-change has to be present for e.g. 10ms before the digital value is updated, it will not be possible to operate the sensor at 200Hz. We have previously assumed that the light sensor is “fast enough”, and we will continue on that course.

Thirdly there are the large rapid changes in external light. These are almost impossible to detect before hand, and they may result in incorrect sorting, but it is possible to do an audit-log, as the

the dynamic calibration can log an error if it determines that the recalibration required is so large that there may have been faulty readings. Then it is only required to manually go through the objects between the last good calibration (incl. no calibration needed) and the audit-log time.

4.3.1.9 Results and Conclusion

Here we will do a quick breakdown of what we have realized in the course of this experiment.

- The use of the flood-light has been disregarded.
- Raw values are just normalized values on a different curve.
- The changes in external light intensity and the intensity of the external light and the location of the external light is highly coupled and is seen as one.
- The distance from the light sensor to the object should be as short as possible.
- The 40W fixed light has to be at least 21cm away from the light sensor in order to stay within the range.
- The stronger the fixed light the less sensitive the measurements are to external changes in light.
- The lower the LUX values are when they change, the larger the change in the readings will be. I.e. choose a weaker fixed light for better reaction to change.
- The NXT light sensor is sensitive “enough” at the higher end of the range.
- The light sensor has a stability of about 20, meaning that the highest accuracy, without running average, is +/- 10. Assuming equal distribution.
- The external light changes may affect the light sensor at optimal conditions with up to 50 reading-points, giving a total maximum inaccuracy of up to 70.
- The light sensor reaction time is per definition “fast enough”.

Or placed in terms of Sensor properties:

- Range: y_min=145, y_max=890, x_min=0LUX, x_max=9410
- Accuracy +/- 70 with no dynamic calibration or running average
- Linearity and offset: offset at least 145, no linearity.
- Resolution: Non-linear, between 1.2 LUX/reading and 25.5 LUX/reading in experiment.
- Reaction time: “fast enough”.

We have also discovered that if the fixed light source is strong enough we can easily recognise non-transparent objects, even if the external light source changes (within normal operating parameters).With semi-transparent it is a little more iffy.

With these experiments it can be concluded that the NXT light sensor is adequate for transparency measurements in the context of this assignment.

4.4 The conveyor belt motor

As described earlier the conveyer belt motor is operated at maximum speed and it is therefore not necessary to perform any experiments.

4.4.1 Result and Conclusion

N/A

5 Conclusion

Based on the test and experiments of the individual components we conclude that the NXT platform with the lejOS can be used for implementing the Automatic sorting conveyer.

This is based on the following observations:

- The NXT light sensor is accurate enough to detect non-transparent objects under the following conditions:
    - The fixed light source is approx. 9410 LUX at the light sensor.
    - A running average sampling scheme is used
    - The measuring area is shielded from extreme external light changes.
    - The objects on the conveyer belt pass within 5 cm of the light sensor.
- The NXT motor is accurate and fast enough to be used as a deflector arm under the following conditions:
    - The battery must be fully charged
    - The objects on the conveyer belt must be placed at an acceptable distance to each other.
- The NXT motor is stable enough to be used as driver for the conveyer belt under the following conditions:
    - The battery must be fully charged

In order to control the determinism of the execution we recommend using the Cyclic Executive scheduling.

This will affect the way the touch sensor is monitored and the polling should be performed from the application and not using the lejOS build-in event listener.

If it is necessary to add speed control to the conveyer belt motor this control should also be implemented as part of the Cyclic executive schedule in the application and not using the build-in speed control.

Appendix

6 Program

6.1 Touch Sensor

6.1.1 EntryPoint Class (Main)

```
package gis.group3.touchsensor.analysis;
import lejos.nxt.Button;
import lejos.nxt.SensorPort;
import lejos.nxt.TouchSensor;

public class EntryPoint {
	public static void main(String[] args) throws InterruptedException {
		TouchSensor ts = new TouchSensor(SensorPort.S1);
		SensorPort.S1.addSensorPortListener(new TouchSensorListener(ts));
		Button.ESCAPE.waitForPressAndRelease();
	}
}
```

6.1.2 TouchSensorListener Class

```
package gis.group3.touchsensor.analysis;
import lejos.nxt.LCD;
import lejos.nxt.SensorPort;
import lejos.nxt.SensorPortListener;
import lejos.nxt.TouchSensor;

public class TouchSensorListener implements SensorPortListener
{
	public TouchSensorListener(TouchSensor ts)
	{
		mts = ts;
	}
	public void stateChanged(SensorPort port, int oldValue, int newValue){
		if (mts.isPressed())
		{
			LCD.drawString("Touch pressed", 1, 2);
		}
		else
		{
			LCD.clear();
		}
		LCD.refresh();
	}
	TouchSensor mts;
}
```

6.1.3 Light sensorMeasurements class

```
package gis.group3.lightsensor.analysis;

import lejos.nxt.*;

public class LightSensorMeasurements {
	public LightSensorMeasurements(LightSensor sensor)
	{
		mSensor = sensor;
		mSensor.setFloodlight(false);
	}

	public static void main(String[] args) {
		LightSensor ls = new LightSensor(SensorPort.S1);
		LightSensorMeasurements lsm = new LightSensorMeasurements(ls);
		lsm.runTestMenu();
	}

	private static final int OK_BUTTON = 0x01;
	private static final int LEFT_BUTTON = 0x02;
	private static final int RIGHT_BUTTON = 0x04;
	private static final int ESCAPE_BUTTON = 0x08;

	public void runTestMenu()
	{
		boolean terminate = false;
```

```
int selection = 2;
while (!terminate)
{
	LCD.clear();
	LCD.drawString("Test menu: ", 0, 0);
	LCD.drawString("Configure", 1, 2);
	LCD.drawString("Reaction time", 1, 3);
	LCD.drawString("Sampling", 1, 4);
	LCD.drawString("Stability", 1, 5);
	LCD.drawString("Terminate app", 1, 6);
	LCD.drawString(">", 0, selection);
	int button = Button.waitForPress();
	if(button == LEFT_BUTTON)
	{
		--selection;
		if (selection < 2)
		{
			selection = 6;
		}
	}
	else if (button == RIGHT_BUTTON)
	{
		++selection;
		if (selection > 6)
		{
			selection = 2;
		}
	}
	else if (button == OK_BUTTON)
	{
		switch (selection)
		{
		case 2:
			calibrate();
			runSampleTest("Configuration");
			break;
		case 3:
			runVariansTest("Reaction");
			break;
```

```
				case 4:
					runSampleTest("Sampling");
					break;
				case 5:
					runVariansTest("Stability");
					break;
				case 6:
					terminate = true;
					break;
				}
			}
		}
	}

	private void calibrate()
	{
		LCD.clear();
		LCD.drawString("Calibration: ", 0, 0);
		LCD.drawString("* Min light", 2, 2);
		LCD.drawString("Press any key", 0, 4);
		LCD.drawString("when ready", 0, 5);
		Button.waitForPress();
		LCD.clear();
		LCD.drawString("Calibration: ", 0, 0);
		LCD.drawString("Calibra. low", 2, 2);
		mSensor.calibrateLow();
		try {
			Thread.sleep(1000);
		} catch (InterruptedException e) {
		}
		int Ndark = mSensor.readNormalizedValue();
		int dark = mSensor.getLow();
		LCD.clear();
		LCD.drawString("Calibration: ", 0, 0);
		LCD.drawString("* Max light", 2, 2);
		LCD.drawString("Press any key", 0, 4);
		LCD.drawString("when ready", 0, 5);
		Button.waitForPress();
		LCD.clear();
		LCD.drawString("Calibration: ", 0, 0);
		LCD.drawString("Calibra. high", 2, 2);
		mSensor.calibrateHigh();
		try {
			Thread.sleep(1000);
		} catch (InterruptedException e) {
		}
		int Nlight = mSensor.readNormalizedValue();
		int light = mSensor.getHigh();
		LCD.clear();
		LCD.drawString("Calibration: ", 0, 0);
		LCD.drawString("Raw low: ", 2, 2);
		LCD.drawInt(dark, 12, 2);
		LCD.drawString("N low: ", 2, 3);
		LCD.drawInt(Ndark, 12, 3);
		LCD.drawString("Raw high: ", 2, 4);
		LCD.drawInt(light, 12, 4);
		LCD.drawString("N high: ", 2, 5);
		LCD.drawInt(Nlight, 12, 5);
		LCD.drawString("Press any key", 0, 7);
		Button.waitForPress();
	}

	private void runSampleTest(String testType)
	{
		boolean terminate = false;
		int selection = 2;
		while (!terminate)
		{
			LCD.clear();
			LCD.drawString(testType + " test: ", 0, 0);
			LCD.drawString("Sample", 1, 2);
			LCD.drawString("Done", 1, 3);
			LCD.drawString(">", 0, selection);
			int button = Button.waitForPress();
			if(button == LEFT_BUTTON)
			{
				--selection;
				if (selection < 2)
```

```
					{
						selection = 3;
					}
				}
				else if (button == RIGHT_BUTTON)
				{
					++selection;
					if (selection > 3)
					{
						selection = 2;
					}
				}
				else if (button == OK_BUTTON)
				{
					switch (selection)
					{
					case 2:
						testRawNorm();
						break;
					case 3:
						terminate = true;
						break;
					}
				}
			}
		}


		private void testRawNorm()
		{
			LCD.clear();
			LCD.drawString("Sampling: ", 0, 0);
			LCD.drawString("* prepare", 2, 2);
		  LCD.drawString("setup", 4, 3);
			LCD.drawString("Press any key", 0, 5);
			LCD.drawString("when ready", 0, 6);
			int light = mSensor.readNormalizedValue();
			int raw = mSensor.readValue();
			LCD.clear();
		LCD.drawString("Sampling: ", 0, 0);
		LCD.drawString("Norm: ", 2, 2);
		LCD.drawInt(light, 8, 2);
		LCD.drawString("Raw: ", 2, 3);
		LCD.drawInt(raw, 8, 3);
		LCD.drawString("Press any key", 0, 5);
		Button.waitForPress();
	}

	private void runReactionTimeTest()
	{
		LCD.clear();
		LCD.drawString("Reaction test:", 0, 0);
		LCD.drawString("* Enable rapid", 2, 2);
	  LCD.drawString("changing", 4, 3);
	  LCD.drawString("light", 4, 4);
		LCD.drawString("Press any key", 0, 6);
		LCD.drawString("when ready", 0, 7);
		Button.waitForPress();

		int[] samples = new int[10000];
		for (int i = 0; i < 10000; ++i)
		{
			samples[i] = mSensor.readNormalizedValue();
		}
		int[] identical = new int[5];
		for (int i2 = 5; i2 < 10000; ++i2)
		{
			for (int i3 = 1; i3 <= 5; ++i3)
			{
				if (samples[i2] != samples[i2 - i3])
				{
					break;
				}
				++identical[i3 - 1];
			}
		}
		LCD.clear();
		LCD.drawString("Reaction test", 0, 0);
```

```
            LCD.drawString("Measured 10000", 2, 2);
            LCD.drawString("Clusters:", 2, 3);
            LCD.drawString("5 sample: ", 2, 3);
            LCD.drawInt(identical[4], 12, 3);
            LCD.drawString("4 sample: ", 2, 4);
            LCD.drawInt(identical[3], 12, 4);
            LCD.drawString("3 sample: ", 2, 5);
            LCD.drawInt(identical[2], 12, 5);
            LCD.drawString("2 sample: ", 2, 6);
            LCD.drawInt(identical[1], 12, 6);
            LCD.drawString("1 sample: ", 2, 7);
            LCD.drawInt(identical[0], 12, 7);
            Button.waitForPress();
    }

    private void runVariansTest(String test) {
            LCD.clear();
            LCD.drawString(test + " test:", 0, 0);
            LCD.drawString("* setup in", 2, 2);
       LCD.drawString("unchanging", 4, 3);
       LCD.drawString("environment", 4, 4);
            LCD.drawString("Press any key", 0, 6);
            LCD.drawString("when ready", 0, 7);
            Button.waitForPress();

            LCD.clear();
            LCD.drawString(test + " test:", 0, 0);
            LCD.drawString("Sampling", 2, 2);
            long start = System.currentTimeMillis();
            int[] samples = new int[10000];
            for (int i = 0; i < 10000; ++i)
            {
                    samples[i] = mSensor.readNormalizedValue();
            }
            long done = System.currentTimeMillis();

            LCD.clear();
            LCD.drawString(test + " test:", 0, 0);
            LCD.drawString("Calculating", 2, 2);
```

```
LCD.drawString("Please wait...", 2, 4);
// Identify the samples with matching neighbors
int[] identical = new int[5];
for (int i2 = 5; i2 < 10000; ++i2)
{
        for (int i3 = 1; i3 <= 5; ++i3)
        {
                if (samples[i2] != samples[i2 - i3])
                {
                        break;
                }
                ++identical[i3 - 1];
        }
}
// Find max and min
// find the average
// Find the varians
int max = samples[0];
int min = samples[0];
long average = 0;
int[] varians = new int[1023];
for (int i2 = 0; i2 < 1023; ++i2)
{
        varians[i2] = 0;
}

for (int i2 = 1; i2 < 10000; ++i2)
{
        average += samples[i2];
        if (samples[i2] < min)
        {
                min = samples[i2];
        }
        if (samples[i2] > max)
        {
                max = samples[i2];
        }
        ++varians[samples[i2]];
}
```

```
average /= 10000;
LCD.clear();
LCD.drawString(test + " test", 0, 0);
LCD.drawString("Measured 10000", 2, 2);
LCD.drawString("samples", 2, 3);
LCD.drawString("in ", 2, 4);
LCD.drawInt((int)(done - start), 6, 4);
LCD.drawString("ms", 9, 4);
LCD.drawString("Press any key", 0, 6);
Button.waitForPress();

LCD.clear();
LCD.drawString(test + " test", 0, 0);
LCD.drawString("High val: ", 2, 2);
LCD.drawInt(max, 12, 2);
LCD.drawString("Low val: ", 2, 3);
LCD.drawInt(min, 12, 3);
LCD.drawString("Average: ", 2, 4);
LCD.drawInt((int)average, 12, 4);
LCD.drawString("Press any key", 0, 6);
Button.waitForPress();

LCD.clear();
LCD.drawString(test + " test", 0, 0);
LCD.drawString("Varians: ", 2, 2);
int index = 0;
for (int i = 0; i < 1023; ++i)
{
        if (varians[i] != 0)
        {
                LCD.drawInt(i, 4, index + 3);
                LCD.drawString(": ", 4 + 3, index + 3);
                LCD.drawInt(varians[i], 4 + 3 + 2, index + 3);
                ++index;
                if (index > 2)
                {
                        LCD.drawString("Press any key", 0, 7);
                        Button.waitForPress();
                        LCD.clear();
```

```
                                LCD.drawString(test + " test", 0, 0);
                                LCD.drawString("Varians: ", 2, 2);
                                index = 0;
                        }
                }
            }
            LCD.drawString("Press any key", 0, index + 3 + 2);
            Button.waitForPress();

            LCD.clear();
            LCD.drawString(test + " test", 0, 0);
            LCD.drawString("Clusters:", 2, 2);
            LCD.drawString("5 sample: ", 2, 3);
            LCD.drawInt(identical[4], 12, 3);
            LCD.drawString("4 sample: ", 2, 4);
            LCD.drawInt(identical[3], 12, 4);
            LCD.drawString("3 sample: ", 2, 5);
            LCD.drawInt(identical[2], 12, 5);
            LCD.drawString("2 sample: ", 2, 6);
            LCD.drawInt(identical[1], 12, 6);
            LCD.drawString("1 sample: ", 2, 7);
            LCD.drawInt(identical[0], 12, 7);
            Button.waitForPress();
        }

        private LightSensor mSensor;
}
```

6.1.4 The DeflectorTest class

```
import lejos.nxt.*;

public class DeflectorTest
{
  public static void main(String[] args)
  {
    int count = 0;

    LCD.drawString("Inertia Test", 0, 0);
    Button.waitForPress();
    Motor.A.setSpeed(1020); // 1020 degrees/sec ~ 170rpm @ 9V

    while (Button.waitForPress() != 8) // End the program when ESCAPE is hit
    {
      Motor.A.resetTachoCount();
      Motor.A.rotate(45,false);  // block and rotate 45 degrees clockwise
      count = Motor.A.getTachoCount();
      LCD.drawInt(count,0,1);
      Button.waitForPress();

      Motor.A.resetTachoCount();
      Motor.A.rotate(-45,false); // block and rotate 45 degrees counter clockwise
      count = Motor.A.getTachoCount();
      LCD.drawInt(count,0,2);
    }
  }
}
```

7 Reference

[1]	http://www.rafa.dk/mleenheder.6
[2]	http://home6.inet.tele.dk/haagen/lystab.htm
[3]	http://www.cs.aau.dk/~apr/ITVHSI/
[4]	http://www.philohome.com/motors/motorcomp.htm