University of Kent

AVINSoR (A Visually Intelligent Navigation System for application in Robotics)

Project REPORT

Gurtej Singh Birring

gsb4@kent.ac.uk

Dr. Peter Lee

Date of Submission: 09/04/2014

# Introduction

*Aim:* The purpose of AVINSoR project is to develop an AGV[[1]](#footnote-1) platform such that allows the user to “teach” the robotic apparatus how to *move* or *actuate* towards a perceived ‘*object of interest’*, appropriately*.* Perceptions, movement patterns, and the *‘cognitive’* decision-making process in between, should be trained by the user rather than hardcoded by any means. The user should be able to *‘define’* and *‘refine’ perception, cognition and motor (movement) patterns* in order to induce the correct behavior from the robot.

*Methodology:* As described in previous AVINSoR documentations, the operations of this project are planned in accordance to an iterative cyclic development model – specifically, the *rapid prototyping* approach is utilized (see Appendix for details). Thus during the course of this project, two prototype are to be actualized; the later building upon the performance and architecture of the former (with due regards to Prototype 1’s evaluation process).

Though the prototypes themselves will be hardware-specific i.e. developed in order to operate with a particular hardware framework in mind, the concept developed will extend beyond the particular hardware framework used to demonstrate the architecture. Though, that being said, implementation of the concept on the specific hardware framework *will in itself* become a major part of the project.

The objectives contrived are based upon the expectations of producing the following deliverables for demonstration/viva-voce examination:

* **Graphical User Interface.**
* **Source Code Library.**
* **Hardware Assembly.**

Objectives, in particular:

* Produce a literature review document covering:
  + Intelligent and adaptive control systems.
    - Human cognition.
      * Off-line simulation of psychological models.
    - Machine learning (pattern recognition).
    - Artificial intelligence.
  + Autonomous Navigation.
    - Image processing.
    - Image analysis techniques.
      * Object Detection.
      * Object Tracking.
* Develop a description of the technical concept. Describe the functionality of the system and the various sub-systems (recognized at this stage).
* Describe the most suitable technical approach, explaining how the rival approaches are ruled out. Elaborate on executive strategy and project management.   
  A *presentation* will be produced whereby the background, system functionality, and (in limited detail) technical approach is explained.  
  An *interim* *report* will cover all the above in greater detail, inclusive of all referencing.
* Establish a working connection

Background; mentioning artificial intelligence, psychological models of cognition (and computer simulations of), biology of cognition (neural connections, etc.), biology of vision, psychology of visual perception.2

Fundamental Concept

The fundamental concept of AVINSoR revolves around the biological, psychological, *ecological* understanding of the visual and sensory perception phenomena, particularly in humans and *similar* (related) mammals.

Vision

## Aims and Objectives

## Background



**Figure x.** A diagram depicting the proposed robot assembly for Prototype 1, and the software required for the functioning of the system on a Windows® PC.

# Prototype 1

Purpose of Prototype 1: To demonstrate Bluetooth

## System Description (Design)

### Proposed Assembly

All *‘cognitive’* processing (i.e. processing with regards to ‘*the cognition system’* viz. intelligent navigation and decision-making) will be accomplished solely on PC using the software set-up depicted in figure 1. The PC is the *client* as it executes the *client* application (*namely* ‘AVINSoR client’) whose function is to controlall other aspects of the system through commands (hardware *and* software).

The robot assembly simply acts as multifaceted *server* – providing sensory input (some of which are processed through the NXT brick) and allowing control of the actuator motors (through NXT brick’s signal processing capabilities). The operation of the NXT brick is configured and instructed through commands. [[2]](#footnote-2)

The purpose of Prototype 1 is to **(1)** assemble the robot and establish connectivity and **(2)** develop the fundamental application (AVINSoR Client) to a reasonable standard so that (further) development of the *‘cognitive’* processes can be pursued with ease in later prototypes. The AVINSoR Client will be developed with maximum *modularity* and *reusability* in mind, effective use of object-orientation will naturally allow easily identifiable code-objects/software-components designed for minimal coupling and maximum cohesion.

#### Hardware

With regards to hardware, nothing is developed. The LEGO® Mindstorms® NXT Kit is a modular hardware kit that will allow the control of attached motors, and the reception of digital data from the attached sensors, with absolutely no hardware development.

The webcam connection to the PC is completely independent of the NXT. The possibility of implementing wireless connectivity between the webcam and the PC was omitted for a later prototype – as time was prioritized for the development of AVINSoR Client – which will simply be referred to as the *software application* or the *client application* throughout this report).

The **sensors** of the AVINSoR Prototype 1 architecture are:

* The Webcam – visual data(intensity of red light, green light and blue light - per pixel)*. [[3]](#footnote-3)*
* The Ultrasonic Sensor – distance of objects.
* The Light Sensor – light intensity.
* The Sound Sensor – sound pressure.

The description associated with each sensor’s purpose is very basic and over-simplified. Ultimately, many things can be deduced from the data received from each sensor, especially when data is *correlated with other sensors* through a *trainable* pattern recognition algorithm.

Note that each sensor may have an angle limit, distance limit, or finite coverage area, out of which the sensor will *not* respond to *detect* the stimuli (due to obvious physical and mechanical reasons). This is the case with humans and animals as much as it is with electronic robots.

The **actuators** of the architecture are:

* Left (Servo) Motor  
  - controls the rotation of the left wheel.
* Right (Servo) Motor  
  - controls the rotation of the right wheel.

#### Software

**Drivers** willallow the operating system (Microsoft® Windows®) to detect and communicate with the hardware.Without the correct compatible drivers, hardware will not be recognized by the operating system (whose function is to provide *hardware abstraction*), and therefore applications will not be able to communicate with the hardware. Thankfully, all necessary drivers to communicate with the NXT Brick via Bluetooth® are provided by the manufacturer (LEGO®) and Microsoft® Windows®. The necessary drivers to communicate with the webcam are provided by the manufacturer (Logitech®) and Microsoft® (UVC webcam drivers are typically innate in Microsoft® Windows® installations).

**MATLAB** is a numerical computing environment widely used by engineers and scientists.

##### AVINSoR Client

This is fundamentally the entire software aspect of the AVINSoR system packaged into a single application.

The **client software** functions to:

* Establish communication with the NXT in order to send commands and receive data.
* Provide a GUI (Graphical User Interface):
  + *Get input* from the human user, in order to:
    1. Train the system to deduce useful information from the array of sensors (digital sensor data).
    2. Train the system to produce precise movement patterns.
  + *Show data* so that user is able to make smarter decisions with regards to *training* the robot. This feature is also *essential* for *testing* and *diagnosis* - during development *and* when in use by the end-user. Such data will include:
    1. Polled *sensor data*, and other NXT variables such as battery level.
    2. The *state* or *result* of any *‘cognitive’* process. This typically includes any machine learning algorithm for the purpose of emulating cognition in humans. This pattern classification, decision-making, object detection/recognition/tracking algorithm (wether *supervised, unsupervised or reinforced)*.
* Communicate with an instance of MATLAB® in order to initiate any complex mathematical processes, or to utilize the functionality of many useful toolboxes and libraries that may be available. This will involve:
  + *Execution* of functions through commands.
  + Transfer of data over to the server.  
    Passing-over of any relevant data to the server, *typically* through the reconstruction of data via (potentially iterative) formulation and execution of command strings.
  + *Retrieval of data from the server.*

#### Robot Assembly

##### Logitech® C270HD Webcam

|  |
| --- |
| http://www.logitech.com/assets/30032/2/logitech-hd-webcam-c260.png |
| Image x. The Logitech C270 webcam. [6] |

The webcam’s function is to provide visual input to the *cognition system*. In Prototype 1 this is a snapshot (i.e. a single image, captured on request). However, in later prototypes, when perception and cognition systems, as well as I/O communication with the NXT assembly has been refined to a standard capable of real-time processing, a continuous 30fps video stream may be used.

|  |  |
| --- | --- |
| Category | Specification |
| Connectivity | Hi-Speed USB 2.0 |
| Resolutions |  |
| Maximum Frame Rate |  |
|  | 30 Frames Per Second |

List of resolutions Table x. Summary of the Logitech® C270’s specification. [6]

**The Logitech C270HD was specifically sourced (over often cheaper alternatives) for the following reasons:** [1]

1. Logitech’s reputation to abide by the UVC device-class specifications (laid out by the USB committee in the relevant USB 2.0 documentation) in order to allow uniformity in communication and hence, compatibility with different software applications. REF.  
     
   This will allow us to capture frames directly from MATLAB®, provided that *MATLAB Support Package for USB webcams* is installed.
2. Logitech’s reputation to provide DirectShow® compliant WDM (Windows® Driver Model) drivers. This is a prerequisite if the device is to be used with MATLAB’s Image Acquisition Toolbox™.   
     
   The Image Acquisition Toolbox™ is a widely used toolbox, its functions often utilized by other MATLAB® libraries and toolboxes (particularly those *not* authored by MathWorks®). We are particularly interested in libraries that allow image processing and analysis. [2]  
     
   DirectShow® is essentially an API that allows applications to easily implement the functionality of high-quality AV capture and feedback. [[4]](#footnote-4) Apart from official DirectShow samples from Microsoft®, many open-source software libraries exist which further encapsulate the DirectShow API to allow an *even more effortless* method of implementing AV functionality into applications through many different programming languages and environments. One such library of interest to our C# client application may be DirectShow.NET – in case MATLAB® proves to be unsuitable, or capturing directly through the client application proves to be more effective for a given purpose. [3] [4] [5]  
     
   Thus, for the purpose of *versatility in development* and *upgradability* – especially as the course of rapid prototyping continues to improve upon this project, DirectShow® compatibility cannot be considered an option.

##### NXT Kit Assembly

The advantage of using the NXT framework over, say, a development board or a single board computer, is that it eliminates the need for creating custom hardware/software interfacing that would be necessary to control the actuators (servo motors), and get readings from various different types of analogue and digital sensors. Such tasks may sound trivial but developing a system that would function to the *equivalent* specification mentioned above (under ‘The LEGO® Mindstorms NXT 1.0 Kit’) *and* provide the equivalent level of encapsulation and concealment of low level hardware and software detail so that focus may be retained on the development of the *cognition* aspects (the *main* scope of this project), can be considered *unnecessarily risky* in a project with a limited time budget (such as this). To support this statement, it is worth mentioning that even the development of Prototype 1 extended *beyond* the time allocated for it - due to unforeseen technical issues requiring micro-iterations of design, development and testing. (refer to explained time allocation).

Essentially, the **NXT Brick** acts as a software *server* with its own (limited) processing ability. The server functions to interpret commands sent from the PC *(the client)* in order to **(1)** process the control signals necessary to accurately control the *actuators* connected, and **(2)** poll the connected *sensors* and transmit the data back to the client PC (upon request, *or* on specified time interval).

The components labelled ‘Left Wheel’ and ‘Right Wheel’ are **NXT Servo Motor Modules** connected to the Brick’s output ports, in the same manner as indicated. This will allow differential wheel steering in the robot. (See Appendix for different methods of steering in robotic vehicles).

The component labelled ‘Ultrasonic Sensor’ is an **NXT Ultrasonic Sensor** module, this module, through the NXT Brick, essentially produces an 8-bit digital signal ranging from 0 to 255 indicative of the distance (in centimeters) of the object within its range.

The component labelled ‘Light Sensor’ is an **NXT Light Sensor** module, this module, through the NXT brick, produces a normalized digital signal ranging from 0 to 100 (*percent*) indicative of the light intensity captured (within the phototransistors response angle – its response angle is explained later). The light sensor also features a red LED, which can be switched on. When the LED is switched on, the sensor functions to measures *reflected light* – which can be a good indicator of distance from the surface of the object ahead. When the LED is switched off, the sensor functions to measure *ambient light* (from the atmosphere).

The component labelled ‘Sound Sensor’ is an **NXT Sound Sensor** module, this module, through the NXT brick, produces a normalized digital signal ranging from 0 to 100 (*percent*) indicative of sound pressure. The NXT brick can be configured to normalize the signal in accord to either (1) the *‘dBA range’ or* (2) the *‘dB range’*. In the *typical* ‘dB’ mode the NXT Brick produces a signal that is equally sensitive to *all* frequencies (even audible frequencies). In ‘dBA’ mode, the signal produced is representative of only the frequencies sensitive to the human ear. [7] [8]

The sound sensor is capable of measuring sound pressure up to 90 decibels, approximately the equivalent sound intensity of a diesel truck 10 meters away, or a lawn mower close by. [7] [11] [12]

The LEGO® Mindstorms® NXT 1.0 Kit is explained in greater detail under **The LEGO® Mindstorms® NXT 1.0 Kit**.

### The LEGO® Mindstorms® NXT 1.0 Kit

The AVINSoR project utilizes the LEGO® Mindstorms® NXT, a modular programmable robotics kit. The kit will simply be referred to as the *hardware framework* or *the NXT* throughout this report. The relevant technical details of the NXT will be discussed here forth[[5]](#footnote-5).

#### The Brick

The NXT is based around an encapsulated battery-powered processing device called the *‘brick’*, which features a monochrome LCD and 4 tactile buttons*.* *Sensor* and *actuator* modules can be attached to the *NXT* *brick* via the 4 *input ports* and 3 *output ports* (respectively).

|  |
| --- |
| http://cache.lego.com/e/dynamic/is/image/LEGO/9841?$main$ |
| Figure x. The NXT brick. Image courtesy of Lego® shop. |

##### Hardware

|  |
| --- |
|  |
| Figure x. A block diagram overview of the NXT brick, as featured on the Lego® Mindstorms® NXT Hardware Development Kit. |

See appendix x for full schematic of the NXT brick.

###### Processing

As illustrated in figure x, the NXT brick features two microprocessors. A *main processor*, and a *co-processor.*

**Main Processor**

Ultimately controls all *user-specific* functionality. *Executes the Virtual Machine.* Low-level activities are outsourced to the co-processor.

|  |  |
| --- | --- |
| Processor Name: | AT91SAM7S256 |
| Manufacturer: | Atmel® |
| Architecture: | 32-bit ARM® (*featuring* ARM7TDMI *CPU*) |
| Flash: | 256 kilobytes |
| RAM: | 64 kilobytes |
| Clock Speed: | 48 MHz |

Table 1. The basic specifications of the main processor. [1, p. 3]

**Co-Processor**

|  |  |
| --- | --- |
| Processor Name: | ATmega48 |
| Manufacturer: | Atmel® |
| Architecture: | 8-bit Atmel® AVR |
| Flash: | 4 kilobytes |
| RAM: | 512 bytes |
| Clock Speed: | 8 MHz |

Table 2. The basic specifications of the co-processor. [1, p. 3]

The co-processor is dedicated to the following tasks: [1, pp. 17-18]

* Power Management (Controlling distribution of power throughout the board, *including* the main processor).
* Creating PWM output signals for the three motor ports.
* Performing analogue-to-digital conversion on the analogue sensor data that may be present at the four input ports.

**Main Processor – Co-Processor communication**

An I2C bus allows the main-processor to communicate with the co-processor. Note that due to hardware limitations, the main-processor can only act as *master* within the I2C communication setup, and hence it is responsible for *initiating* any transmission or reception activity between itself and the co-processor in order to exchange data. The I2C bus is set up (in both MCUs) to communicate at 380Kbps. [1, p. 18]

Two section of memory are allocated in each microprocessor and updated every 2 milliseconds. This allows both processors to act (largely) independently. Both sections of memory reserved for communication protocol (in each MCU) are representations of a struct in the embedded C programming language: [1, pp. 18-20]

**IOTOAVR** essentially represents the section of memory in each microcontroller reserved for data sent *from* the main processor *to* the coprocessor.

**typedef** struct

**{**

UBYTE Power**;**

UBYTE PwmFreq**;**

SBYTE PwmValue**[**NOS\_OF\_AVR\_OUTPUTS**];**

UBYTE OutputMode**;**

UBYTE InputPower**;**

**}**

IOTOAVR**;**

|  |  |  |
| --- | --- | --- |
| Variable Name | Type (native range) | Description |
| Power | Unsigned Byte  (0 to 255) | Set to ‘0x00’ during normal communication; is only modified during *power down* and *firmware update*. |
| PwmFreq | Unsigned Byte  (0 to 255) | The pulse width modulation *frequency* of the three motor outputs. The LEGO® firmware typically uses a value of 8 (representing 8 KHz), but the acceptable range is from 0 (KHz) to 32 (KHz). |
| PwmValue[2] | Array of  Signed Byte  (-128 to +127) | The pulse width modulation *value* of each of the three motor outputs.   |  |  | | --- | --- | | Array Index (#) | Output | | 0 | Motor A | | 1 | Motor B | | 2 | Motor C |   The acceptable range is from -100 to +100.  The *magnitude* represents the output power intensity (as a *percentage*).  The polarity represents clockwise (-) or anticlockwise (+) rotation. |
| OutputMode | Unsigned Byte  (0 to 255) | This is a bitwise variable.   |  |  | | --- | --- | | Bit # | Output | | 0 | Motor A | | 1 | Motor B | | 2 | Motor C |   A value of ‘0’ indicates that the motor output should be left *floating* when no power is applied (i.e. PWM value = 0). This is referred to as “coasting”.  A value of ‘1’ will indicates that the motor output should be ‘short circuited’, i.e. both motor terminals will be held at ‘1’. This will force a “brake”. This is the *default* setting for all motors, and generally considered to be more accurate (as the motor will stop at a specific point rather than coast to a stop). However, this method *does* consume battery power. [2] |
| InputPower | Unsigned Byte  (0 to 255) | This is a bitwise variable.  The value associated with each sensor indicates *whether* the sensor should be supplied with 9V power, and *how*.   |  |  | | --- | --- | | Bit #s | Output | | 0, 1 | Sensor 1 | | 2, 3 | Sensor 2 | | 4, 5 | Sensor 3 | | 6, 7 | Sensor 4 |   A value of ‘00’ indicates that the sensor should not be supplied with 9v power. (This is the appropriate value if a sensor is *not* connected to the port, if a sensor is *passive*, or the sensor is *self-powered*). [[6]](#footnote-6)  A value of ‘01’ indicates that the 9v power to the sensor should be pulsed, so that power is *not* supplied to the sensor when data is being read from it. Duration: 3 milliseconds supply, 0.1 millisecond measuring time (no 9v supply). This mode is for sensors categorized as ‘*active sensors’*, a legacy category that allow sensors from old LEGO® Mindstorms Robotics Invention System kits to be used with the NXT brick. [1, p. 7]  A value of ‘11’ indicates that the 9v power should *always* be supplied to the sensor. This mode is for use with sensors categorized as *‘digital sensors’* such as the LEGO® NXT kit’s Ultrasonic Sensor. [[7]](#footnote-7) |

**IOFROMAVR** essentially represents the section of memory in each microcontroller reserved for data sent *from* the coprocessor *to* the main processor.

**typedef** struct

**{**

UWORD AdValue**[**NOS\_OF\_AVR\_INPUTS**];**

UWORD Buttons**;**

UWORD Battery**;**

**}**

IOFROMAVR**;**

|  |  |  |
| --- | --- | --- |
| Variable Name | Type (native range) | Description |
| AdValue[3] | Array of  Unsigned Word  (0 to 65535) | The array holds 10-bit values from the analogue-to-digital conversion of the input sensors’ analogue input. The raw values from the ATmega48’s 10-bit ADC converter range from 0 to 1023.   |  |  | | --- | --- | | Array Index (#) | Input | | 0 | Sensor 1 | | 1 | Sensor 2 | | 2 | Sensor 3 | | 3 | Sensor 4 | |
| Buttons | Unsigned Word  (0 to 65535) | Buttons 1, 2 and 3 are connected to the co-processor’s input pin (to an ADC channel – ADC3) via a resistor ladder. Hence the status of all three momentary switches can be deduced from the value of the analogue-to-digital conversion stored *here*.  Button 0 is connected to the PD3/INT1 of the ATmega45 co-processor, a decimal value of 2047 is *added* to *this value* when Button 0 is pressed.  See Keypad schematic in Appendix x, and m\_sched.h |
| Battery | Unsigned Word  (0 to 65535) | This *word* (16-bit value) holds data of **(1)** the measured battery level, **(2)** whether the brick is being powered by 6x AA batteries or an NXT brick battery-pack, *and* **(3)** the version of the co-processor’s firmware.   |  |  |  | | --- | --- | --- | | Bits (#) | Value |  | | 15 | ‘0’ – 6x AA batteries  ‘1’ – Battery Pack | Battery type. | | 13 – 14 | 0 to 3 (number - major version) | Co-processor firmware version. | | 10 – 12 | 0 to 7 (number - minor version) | | 0 - 9 | 0 to 1023 | Gives the battery’s voltage output (in millivolts) when multiplied by 13.848. | |

###### Interface

The NXT Brick features three output ports (for motors), and three input ports (for various types of sensors).

All ports utilize 6-pin modular connection (6P6C RJ12). The brick features *female* connectors, to allow cables extending from motor or sensor modules to be plugged *in* via a *male* connector.

Motor Ports

Table x depicts the signals featured in the 6-pin connection to/from an NXT motor module. Note that the outputs to the DC motors themselves (MA0 and MA1) are sourced from motor drivers U1 (LB1836M) and U2 (LB1930M) controlled by PWM outputs from the ATmega48 co-processor, whilst the digital input pulses from the *incremental rotary encoders* are fed directly to the main processor via a Schmitt trigger that functions to fully recover the signal from the effects of power-attenuation and noise. [[8]](#footnote-8)

|  |  |  |
| --- | --- | --- |
| Pin | | Description |
| # | **Name** |
| 1 | Mx0 | Pulse-width modulated power signal to the motor (pin 1). |
| 2 | Mx1 | Pulse-width modulated power signal to the motor (pin 2). |
| 3 | GND | Ground – shared by all output ports. |
| 4 | POWERMx | 4.3V power from the NXT brick – shared by all input and output ports. Max current of 20mA can be drawn. Short-circuit to GND causes the brick to reset. |
| 5 | TACHOx0 | Inputs to Schmitt trigger circuitry, which leads to the *main processor*. |
| 6 | TACHOx1 | Inputs to Schmitt trigger circuitry, which leads to the *main processor*. |

Table x. P descriptions of the output port. [1, p. 5]

TACHOx0 and TACHOx1 are used by the default LEGO® firmware to capture the two outputs from the incremental rotary encoded embedded within the motor module, and hence, deduce the *direction of rotation* of the motor, and *count the degrees of rotation*. This feature allows us to *limit* the motor rotation (in either direction) to a *specific* number of degrees *in code*. [3, pp. 51-52]

Sensor Ports

LEGO® Mindstorms® NXT recognizes three types of sensors: [1, pp. 6-7]

* Passive Sensors – i.e. sensors that do not have any special timing or measurement requirements. These sensors produce analogue outputs that lead to the *co-processor*, which executes analogue-to-digital conversion every 3mS.
* Digital Sensors – i.e. sensors that have their own microcontroller that handles sampling and any analogue-to-digital conversion necessary. Unlike passive sensors, these sensors use the I2C protocol to communicate *directly* with the *main processor* at 9Kbps. *Note* that the *main processor* is always the *master* in any I2C communication established.
* Active Sensors – *legacy category* to allow sensors from earlier LEGO® Mindstorms kits (specifically LEGO® Mindstorms Robotics Invention System) to be used with the NXT. These sensors produce analogue outputs that lead to the *co-processor*, which executed analogue-to-digital conversion every 3mS.

|  |  |  |
| --- | --- | --- |
| Pin | | Description |
| # | **Name** |
| 1 | ANx | Analogue input from signal. |
| 2 | GND | Ground – shared by all input ports. |
| 3 | GND |
| 4 | IPOWERx | 4.3V power from the NXT brick – shared by all input and output ports. Max current of 20mA can be drawn. Short-circuit to GND causes the brick to reset. |
| 5 | DIGIxI0 | Leads to digital I/O pin of the *main processor.* Typically used for 9.6Kbps I­2C communication with digital sensor by default firmware. May also be used to control aspects of the sensor (e.g. switching on LED in the NXT Light Sensor, s). |
| 6 | DIGIxI1 | Leads to digital I/O pin of the *main processor.* Typically used for 9.6Kbps I­2C communication with digital sensor by default firmware. May also be used to control an aspect of the sensor (e.g. switching on LED in the NXT Light Sensor). |

It may also be of interest that DIGIDI0 and DIGIDI1 (i.e. pins 5 and 6 of Sensor 4) also lead to an ST485 IC (RS485 communication chip) to allow the Sensor 4 port to function as a high-speed communication port – though LEGO® has not developed any devices that utilize this. High-speed communication settings in within default NXT firmware: [1, p. 8]

|  |  |
| --- | --- |
| Specification Criteria | Value |
| Communication Speed | 921.6Kbps |
| Data Bits | 8 bits |
| Stop Bits | 1 bit |
| Parity | 0 bits |

###### PC Communication

The NXT brick features USB and Bluetooth communication which may be used to update the firmware, transmit data or receive commands.

USB

The NXT brick features a full-speed (12Mbps) USB 2.0 communication port. This feature is embedded within the AT91SAM7S256 IC *(main processor)*.

Bluetooth®

The *main processor* connects to a standalone IC (CSR BlueCore™ 4 with external 8Mbit FLASH memory) in order to implement Bluetooth® functionality.

|  |
| --- |
|  |
| Figure x. A diagram illustrating the interface between the *main processor* and the BlueCore™ IC. SPI interface allows the BlueCore™ chip to be updated if ever deemed necessary, and is not used in normal operation. Reset pin re-initializes the IC on start up, and may be used by firmware to disable Bluetooth functionality. BC4-CMD and ARM7-CMD are used to indicate the type of data expected, whilst the UART is used to transfer data and commands between the two ICs. [1, pp. 13-14] |

The main processor may configure the BlueCore™ chip to “stream-mode” in order to transfer data at a baud-rate of 220Kbps (when connection is established), *or* in “command-mode” in order to receive commands to control the Virtual Machine. It is the *later* mode that will allow us control over the motors and sensors via the VM. The IC is configured to function as a Bluetooth® Class II device in order to save power (giving maximum communication range of 10 meters).

BecauseBluetooth® functionality has been implemented via Serial Port Profile (SPP) paradigm (whereby the communication link is considered a ‘serial port’ as far as software is concerned), it becomes very easy to establish a connection to the NXT via different PC software development frameworks / programming languages, as most feature libraries explicitly for serial port communication.

*More information is available in the LEGO® Mindstorms® NXT Hardware Development Kit documentation (pages 12-14, and Appendix 8).*

##### Software°

###### The Virtual Machine

RXE Executables

#### Actuator – Servo Motor Module

Schematic of the NXT motor module can be found in Appendix x.

##### Motor

##### Incremental Rotary Encoder

|  |
| --- |
| http://cache.lego.com/r/www/r/mindstorms/community/-/media/franchises/mindstorms/community/elements/sensors/nxt%20light%20sensor.png?l.r=-1451730025 |
| Figure x. LEGO® Mindstorms® NXT light sensor. Courtesy of Lego Shop. |

#### Sensors – Light Sensor

The light sensor functions to represent light intensity through an analogue output signal.   
This sensor may be configured to detect *ambient light* (i.e. light from the atmosphere with the LED off) or *reflected light* (i.e. light from the red LED reflecting off nearby surfaces facing the sensor, after it is switched on).

The component responsible for converting light intensity into current (analogue signal) is the SFH309 phototransistor, with a response angle of 24° from centre. [4]

The light sensor has no special timing or measurement requirements as it simply outputs an analogue signal and features no microprocessor of its own, hence, it is considered to be a *passive sensor*.

Schematic of the NXT Light Sensor can be found in Appendix x.

#### Sensors – Sound Sensor

|  |
| --- |
| http://cache.lego.com/e/dynamic/is/image/LEGO/9845?$main$ |
| Figure x. LEGO® Mindstorms® NXT light sensor. Courtesy of Lego Shop. |

The sound sensor functions to represent sound intensity through an analogue output signal. The sound sensor is capable

Schematic of the NXT Sound Sensor can be found in Appendix x.

#### Sensors – Ultrasonic Module

Schematic of the NXT Ultrasonic Sensor can be found in Appendix x.

### User Interface (and trainability)

##### GUI design and functionality

## System Implementation

### Setting up a Microsoft Windows® machine for Development and Testing

This section of the report will cover the absolute minimal level of installations required in order to start developing with the Lego® NXT® Brick via a Bluetooth connection.

Initially, the USB drivers that allow the Windows® PC to communicate with the LEGO® Mindstorms® NXT brick, *need* to be installed. Thankfully, this is a “plug and play” operation, whereby as long as the necessary drivers are downloaded and *extracted* from the LEGO® website (The ‘NXT Fantom Driver’), the operation is relatively automatic. The installation of the drivers is triggered by connecting to the NXT brick using the provided Windows® Bluetooth interface. Ensure that Bluetooth has been enabled on your NXT brick, under ‘settings’, and that the device is made *visible*.

A software application such as the Bricx Command Center (open source) or Robomatter, Incorporated’s ROBOTC for Mindstorms can be used to test the operation of the drivers.

#### MATLAB® to Brick connectivity

MathWorks® Matlab® is, unarguably, the most popular software platform across many industries, for the development of numerically-intensive computational systems. The environment features built-in functions that are particularly useful for appropriately manipulating matrices and vectors.

MATLAB® will be particularly useful in the design *and* development phase of AVINSoR, largely due to reduced implementation time of algorithms, and the ability to test developed algorithms with example data relatively easily without the need to develop a GUI[[9]](#footnote-9) - as many useful graphical/geometric plotting functions are innate to MATLAB®. Additionally, and perhaps, more *importantly* MathWorks® provide a number of toolboxes that allow the propagation of numerically-intensive systems such as a number of neural network classifiers (for pattern recognition via fitting, or clustering), fuzzy logic systems, control systems systems, various feature extraction algorithms (particularly for visual/imaging systems), etc.

Though all toolboxes were available for installed MATLAB® version 2014b, the particular toolboxes of interest were: Image Acquisition, Computer Vision, Image Processing, Statistics and Machine Learning Toolbox, and the Neural Network Toolbox.

It is obvious that any mathematical modelling or execution of algorithms is useless if the PC (specifically the software development environment) cannot interact with the Brick to induce calculated and appropriate actuator movement or retrieve live data from the sensors in order to carry out the appropriate calculations. Thus, a means to connect the Brick to MATLAB is necessary. A MATLAB toolbox named ‘RWTH – Mindstorms NXT Toolbox’, developed and maintained by RWTH Aachen University and released under the open source GNU GPL license[[10]](#footnote-10) allows convenient interfacing with the Brick.

Note: It is important that the Instrument Control Toolbox is installed as it is utilized by the RWTH toolbox for Bluetooth communication (the toolbox features built in Bluetooth® serial protocols).

This is achieved by the toolbox through a set of open source C libraries called Libusb which aims to provide ‘easy’ generic access to USB devices. In order to allow the MATLAB® toolbox to successfully connect with the Brick, it is *vital* that ‘inf-wizard’, an application provided by Libusb, is allowed to make the necessary modifications to the original NXT Fantom driver.

Following this, the robot assembly may be tested using MATLAB code provided (see Appendix).

#### C# to Brick connectivity

During the design stage of the project, it was soon realized that MATLAB alone would not *suffice* in the development of a (user-side) software application, largely due to a limited capability GUI designer, and the inability to use external or custom-developed GUI controls. Though, a non-MathWorks® publications was sought, which explained rather well, various means of utilizing Java Swing[[11]](#footnote-11) components *in* GUIDE (MATLAB’s Graphic User Interface Design Environment), as well the ability to integrate Java into the MATLAB programming environment in order to execute various other tasks such as networking and data-processing (largely through code-snippets and examples). [1]

The idea of importing Java functionality *into* MATLAB was ultimately dismissed. Largely due personal limitations in experience with Java, particularly with regards to Java GUI graphing components. There were also concerns about Java VM’s speed of execution, ultimately affecting the re-activity of the robot to transmitted commands, and the visual throughput of the received (sensor) data to the user’s screen.

The case in favor of C#-MATLAB integration relied upon the following:

* MATLAB COM Automation Server (see Appendix) can be easily utilized to execute MATLAB commands and retrieve workspace data. *This feature is not specific to C# or the .NET Framework, and can also be used with any framework/language that supports pointers to a COM Server (and hence act as a client).*
* Previous experience with C#, backend and frontend development.
  + Particularly experience with C# GUI graphing controls.
* .NET Framework significantly (and statistically) faster that Java VM. *Ref here*
* Popularity of C# means greater number of robotics, AI and image processing related libraries are available – therefore the overall functionality of an application that merges .NET Framework functionality with MATLAB functionality is greatly enhanced. Such libraries include AForge.

The MindSqualls library, released free under the GNU GLPU license, is responsible for handling Bluetooth communication between the C# application and the NXT brick, allowing control of actuators, and polling of sensors. No additional driver support or modification is required.

The command line test utility may be used to test the connectivity and robot assembly at this stage.

#### Common Problem: Dysfunctional screen?

A common problem that LEGO® NXT Bricks face is a screen which

### Developing the GUI Interface

### Back-end Code Structure

Useful techniques: Partial classes, multithreading,

Discuss coupling/cohesion – for future improvement.

## Test Results

#### Connectivity

#### Sensor Polling

#### Movement

#### Sensory Pattern Classification

#### Visual Object Recognition

#### Evaluation/Discussion

# Conclusion

## Achievements

## Comparison against Specification

## Future Considerations

How current system can be improved for efficiency and effectiveness.

## Further Development

Further technical developments, building *upon* the specification, and how specs can be achieved.

1. AGV: Autonomous Ground Vehicle. See Appendix for the various categories of autonomous robots. [↑](#footnote-ref-1)
2. Explain client-server. [↑](#footnote-ref-2)
3. See Appendix for RGB, CYMK all that schmuk. [↑](#footnote-ref-3)
4. API an abbreviation for Application Programming Interface. Essentially a software library for the specific purpose of application development.  
   AV an abbreviation for Audio and Video. [↑](#footnote-ref-4)
5. The scope of the AVINSoR project is concerned with the design and implementation of an *intelligent* *autonomous navigation system.* The LEGO® Mindstorms® NXT kit is to *demonstrate* the [↑](#footnote-ref-5)
6. A sensor is *passive* if it does not require power (i.e. LEGO® NXT kit’s Touch Sensor, Light Sensor, Sound Sensor and Temperature Sensor).

   Though LEGO® does not manufacture ­*self-powered* sensors (i.e. sensors that draw power from external power sources), such sensors *may* exist for the NXT, and there is a possibility that such sensors *may* be developedby users (if found necessary for a particular application e.g. where a sensor requires greater power than can be sourced from the NXT’s own power supply; or *if* NXT’s power supply, power management, or *both*, need to be superimposed in a complex application). [↑](#footnote-ref-6)
7. [↑](#footnote-ref-7)
8. See NXT brick hardware schematic in appendix a. [↑](#footnote-ref-8)
9. GUI - Graphical User Interface. [↑](#footnote-ref-9)
10. GNU GPL – GNU General Public License [↑](#footnote-ref-10)
11. Java’s main GUI library. [↑](#footnote-ref-11)