**ECE 578/478 Final Report**

Nixon-Bohr, formerly Bohr

11/19/2013

Jordan Fluth, Vernon Jones



Contents

Contents

[Contents 3](#_Toc374698778)

[Description 5](#_Toc374698779)

[Technical documentation 6](#_Toc374698780)

[Components and subsystems 6](#_Toc374698781)

[Control Motions – defined in the FAAST profile below 7](#_Toc374698782)

[Motor Mode 7](#_Toc374698783)

[Arm Mode 7](#_Toc374698784)

[Hand Mode 8](#_Toc374698785)

[All Modes 9](#_Toc374698786)

[Problems Encountered and their Solutions 9](#_Toc374698787)

[Translating this Knowledge to ECE 479/579 10](#_Toc374698788)

[Technical or Research Problems 10](#_Toc374698789)

[Instructions 10](#_Toc374698790)

[Project Code 12](#_Toc374698791)

[Visual Basic Code: 12](#_Toc374698792)

[FAAST Profile: 12](#_Toc374698793)

[ROBOTC Code: 12](#_Toc374698794)

# Description

Our project modified Bohr to be larger and added software to control the modified robot using a Microsoft Kinect communicating with a Lego NXT via Bluetooth. We got this idea from a youtube video; a demonstration of a master's thesis about controlling a robot using the Kinect. His youtube video can be seen here: <https://www.youtube.com/watch?v=FTBBlt9uk1A>

While we didn't use any of the code he provided (it was under an ambiguous license), we did use some of his ideas in our design.

Our first action was to graft a Halloween robot's skeleton onto the existing Bohr robot. Bohr's existing head was added to the top of the robot, while his existing arm was left at the base. Many of the servos controlling the arm were swapped out; the servo controlling the shoulder horizontal motion was malfunctioning, while the three servos in the hand were continuous-motion, where 180-degree servos were more appropriate. We wrote a suite of programs that leverage existing technology to control the wheels and the arm of the new Nixon-Bohr robot. Our suite includes a Visual Basic script and controller code written for the NXT, leveraging a Microsoft Kinect, Bluetooth, a Kinect gesture-to-keypress program, and a Lego NXT.

The Microsoft Kinect SDK contains a skeleton trace feature that creates a map of the upper half of the body to recognize body positions and motion. These positions and movements are converted into keyboard key presses using FAAST. The generated key presses are input to a Visual Basic script, which transmits the key presses as ASCII characters over Bluetooth to a Lego NXT brick. The NXT brick converts these ASCII characters to motor and servo movements. Using this method we achieve wireless user-defined remote control over the robot.

The motions to control the robot are simple. For instance, for the motor control, height differences between wrists and elbows are used to send forward/back/left/right commands. The arm system is more difficult to control because it has more degrees of freedom than the motor system. We split the arm into shoulder/elbow control, and wrist/hand control to have a more consistent set of motions to use to control the robot.

# Technical documentation

## Components and subsystems

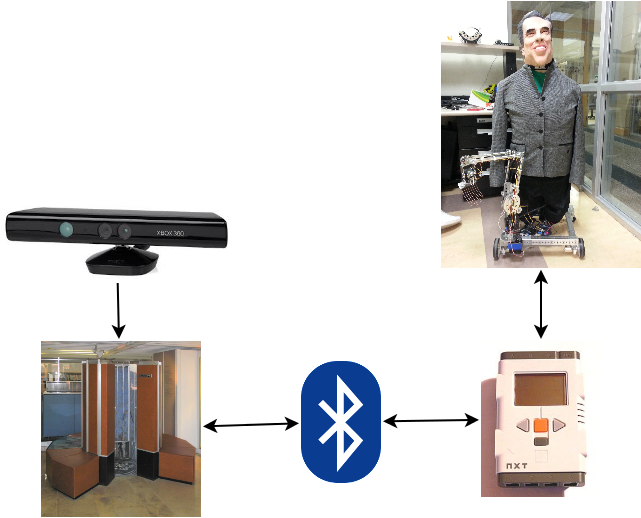


Figure 1: Illustration of Nixon-Bohr components

1. FAAST - Program that translates gestures detected by the Microsoft Kinect into key presses
2. Personal Computer with Bluetooth capability – This is the central hub for receiving and transmitting data. Takes key presses from FAAST and sends keys to the robot over Bluetooth
3. Lego NXT Brick – Provided by the class. The brick runs a RobotC program that translates ASCII character commands sent over a Bluetooth connection into motion.
4. Hi-Technic Motor Controllers (2) – Provided by the class. One of the controllers controls the wheel motion. The other controls shoulder and elbow motion.
5. Hi-Technic Servo Controller – Provided by the class. This controls the motion of the hand, wrist, and left and right rotations of the shoulder.
6. Microsoft Xbox 360 Kinect – Provided by the class. This uses two cameras and a range finder to give the user vision with depth. It also provides a skeleton trace which is needed and used by the FAAST program.

## Control Motions – defined in the FAAST profile below

### Motor Mode

Motor mode includes the two motors that turn the front wheels.

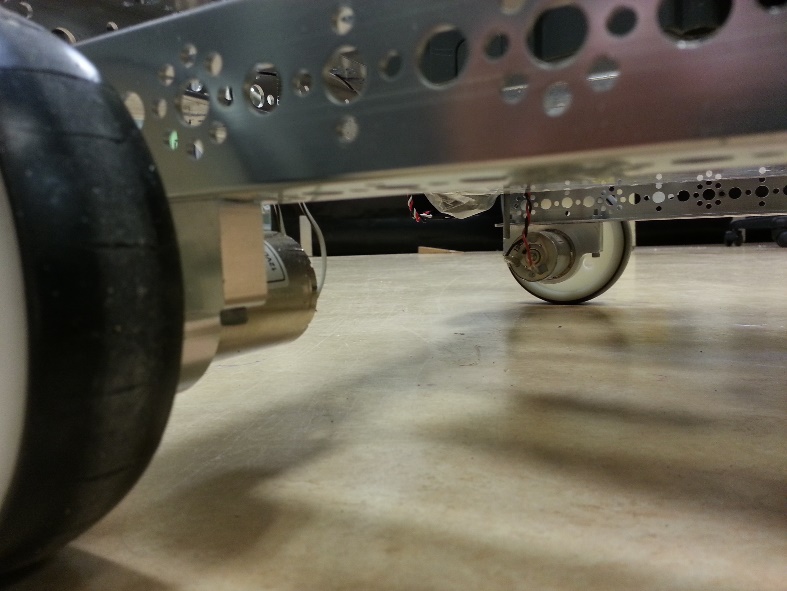


Figure 2: Photo of motors that turn wheels

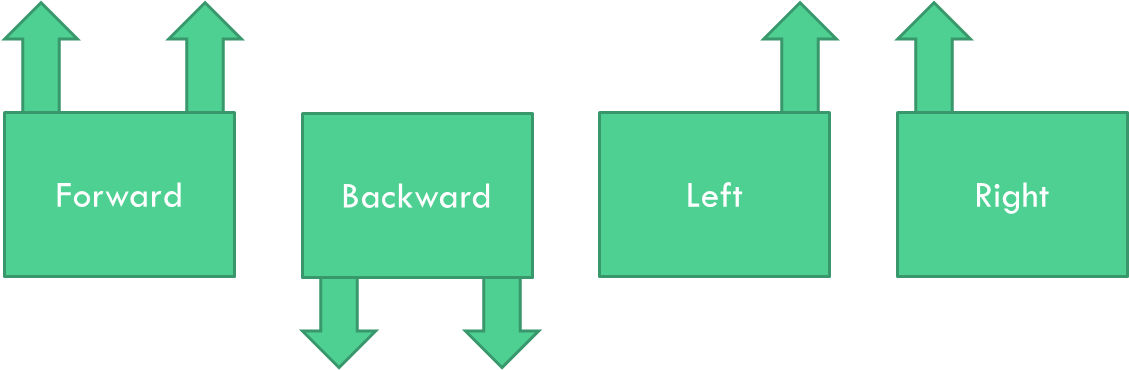


Figure 3: Diagram of motor movements

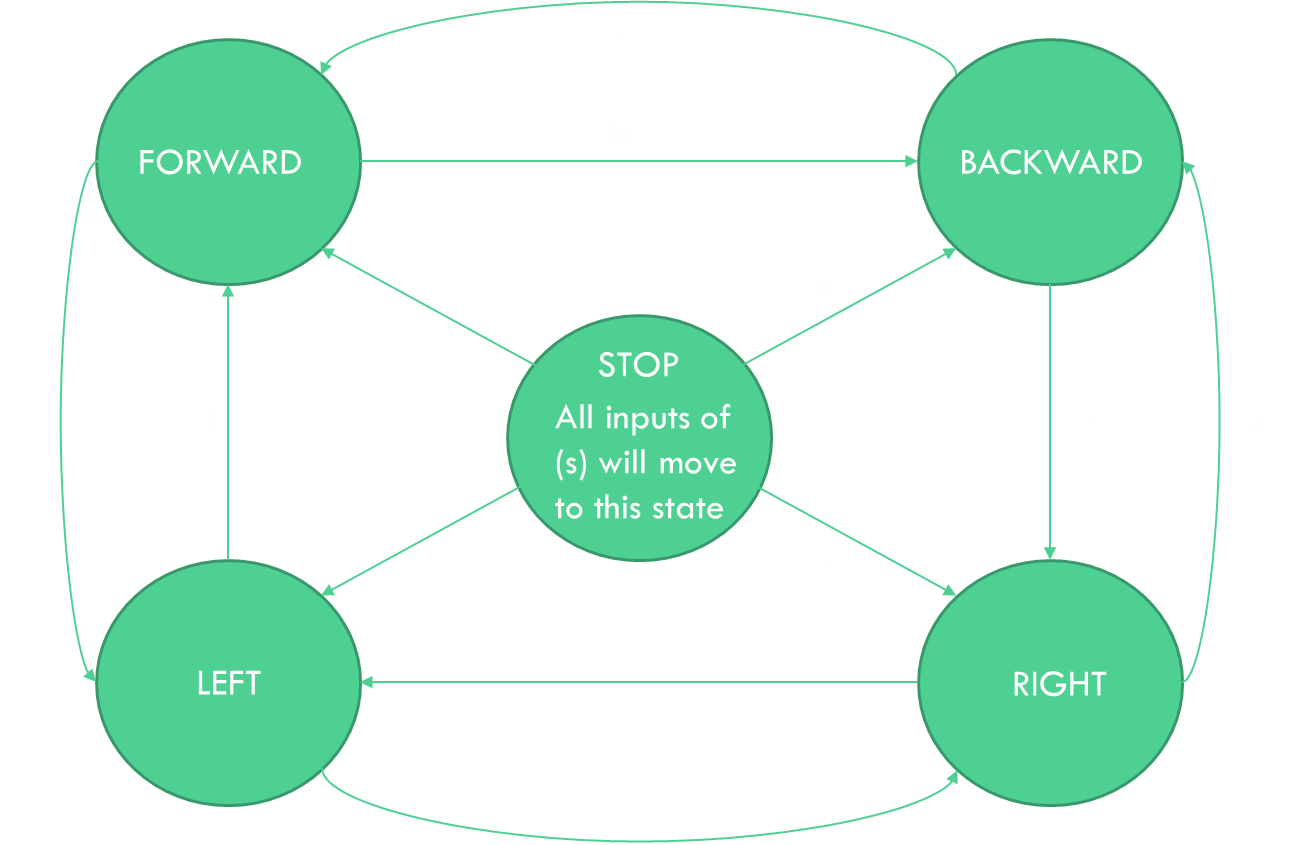


Figure 4: Motor mode state diagam

|  |  |  |
| --- | --- | --- |
| **Action** | **Gesture** | **Associated Key** |
| Switch to Arm/Hand Mode (whichever was last used) | Left hand to the Left, quickly | s |
| Forward | Both wrists above elbows (arms up) | f |
| Backward | Both wrists below elbows (arms down) | b |
| Left | Left wrist above right wrist | l |
| Right | Right wrist above left wrist | r |
| Stop | Both wrists near elbows in height (neutral) | t |

### Arm Mode

Arm mode includes the shoulder servos and the elbow servo.

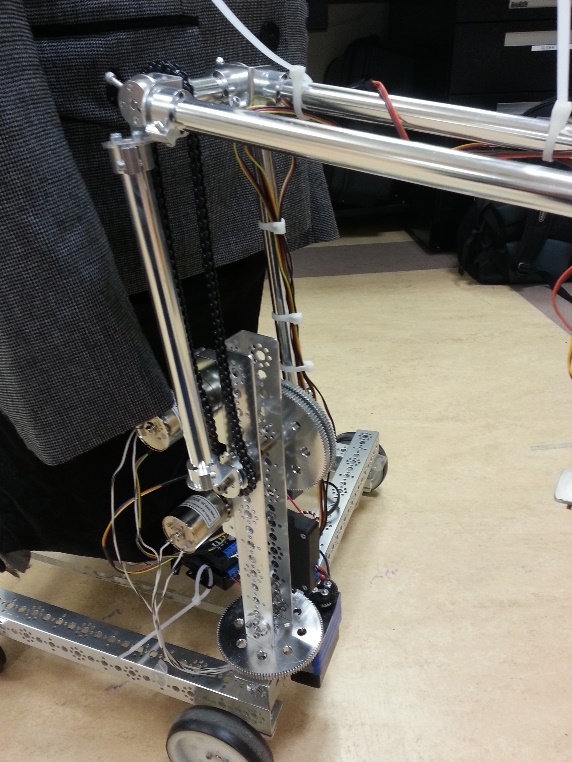


Figure 5: Photo of arm

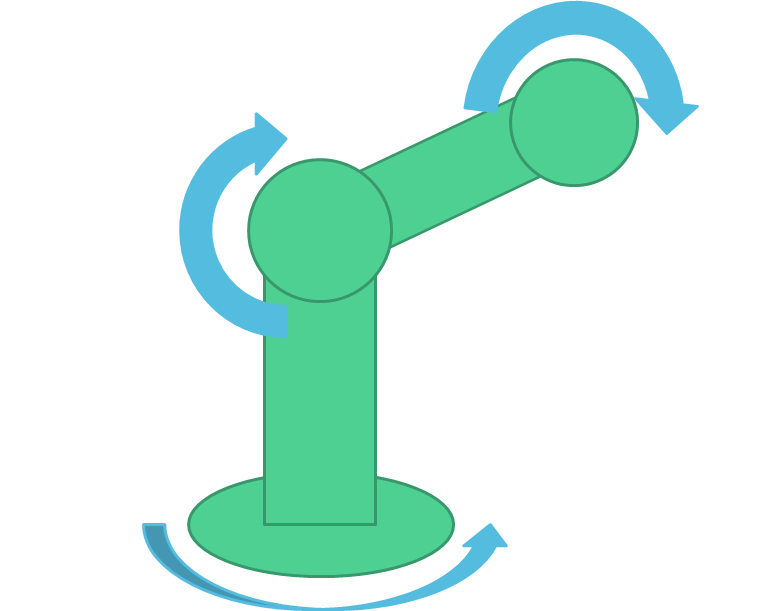


Figure 6: Arm degrees of freedom

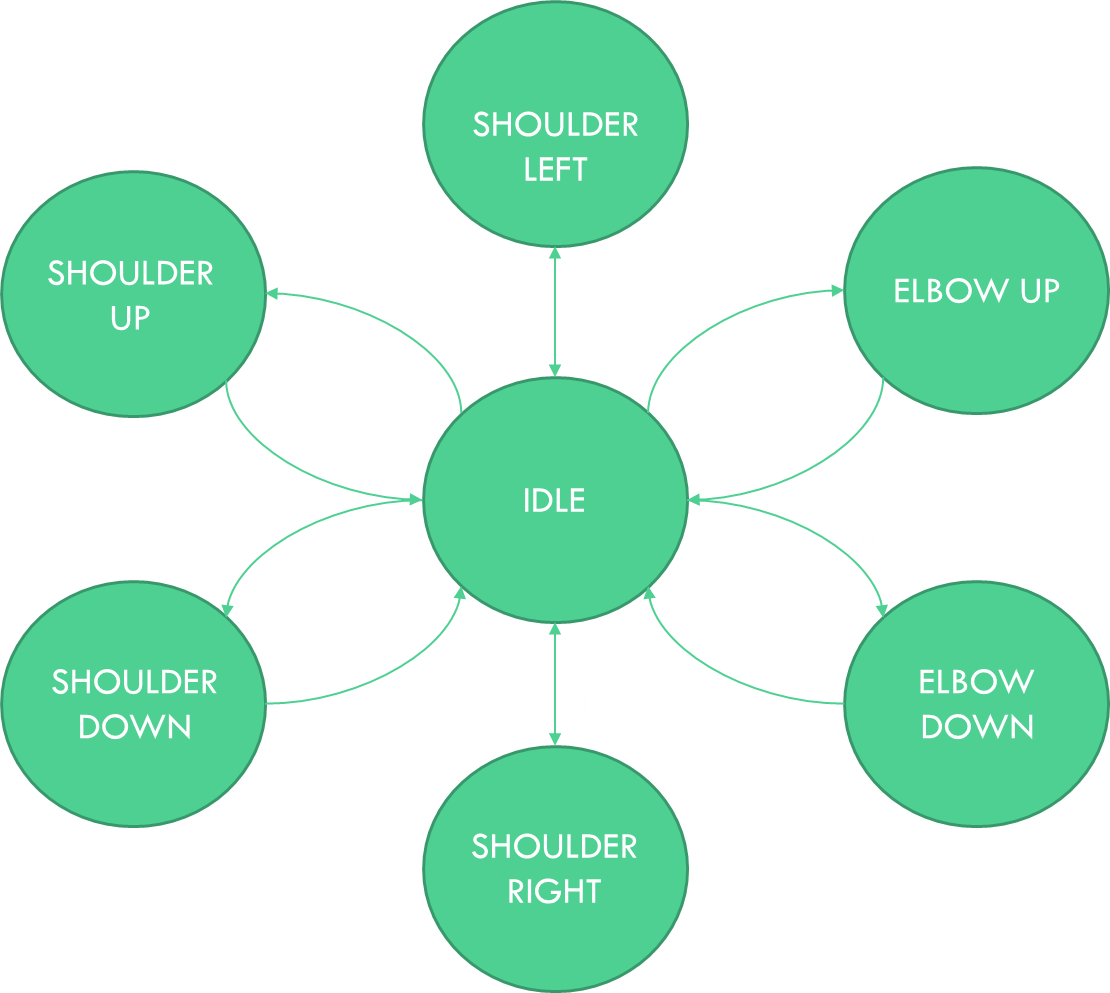


Figure 7: Arm state diagram

|  |  |  |
| --- | --- | --- |
| **Action** | **Gesture** | **Associated Key** |
| Switch to Hand Mode | Left hand to the right, quickly | w |
| Switch to Motor Mode | Left hand to the left, quickly | s |
| Shoulder Left | Right hand to the left | i |
| Shoulder Right | Right hand to the right | j |
| Shoulder Up | Right hand up | z |
| Shoulder Down | Right hand down | x |
| Elbow Up | Left hand up | k |
| Elbow Down | Left hand down | m |

### Hand Mode

Hand mode includes the wrist servos and the servo responsible for opening and closing the hand

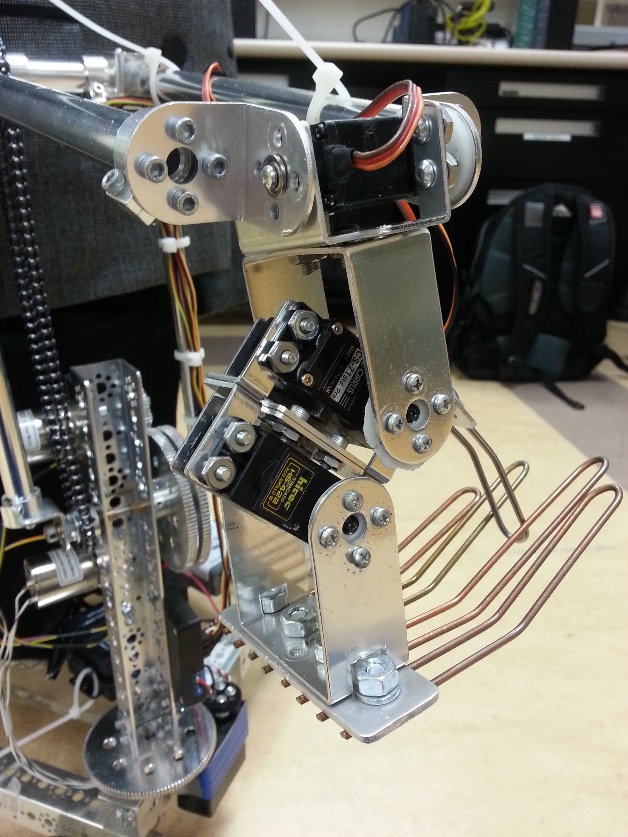


Figure 8: Photo of hand

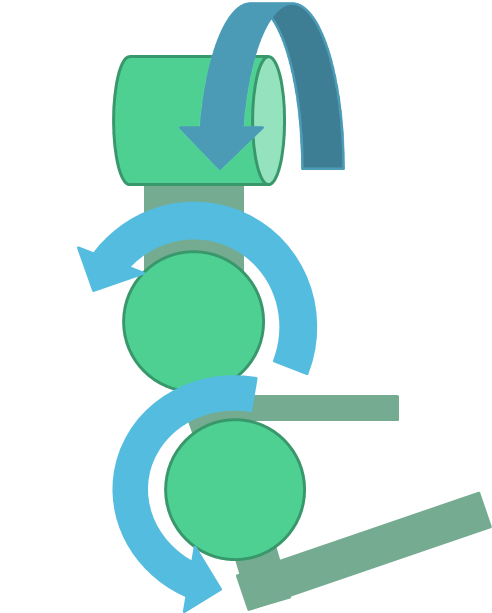


Figure 9: Hand degrees of freedom diagram

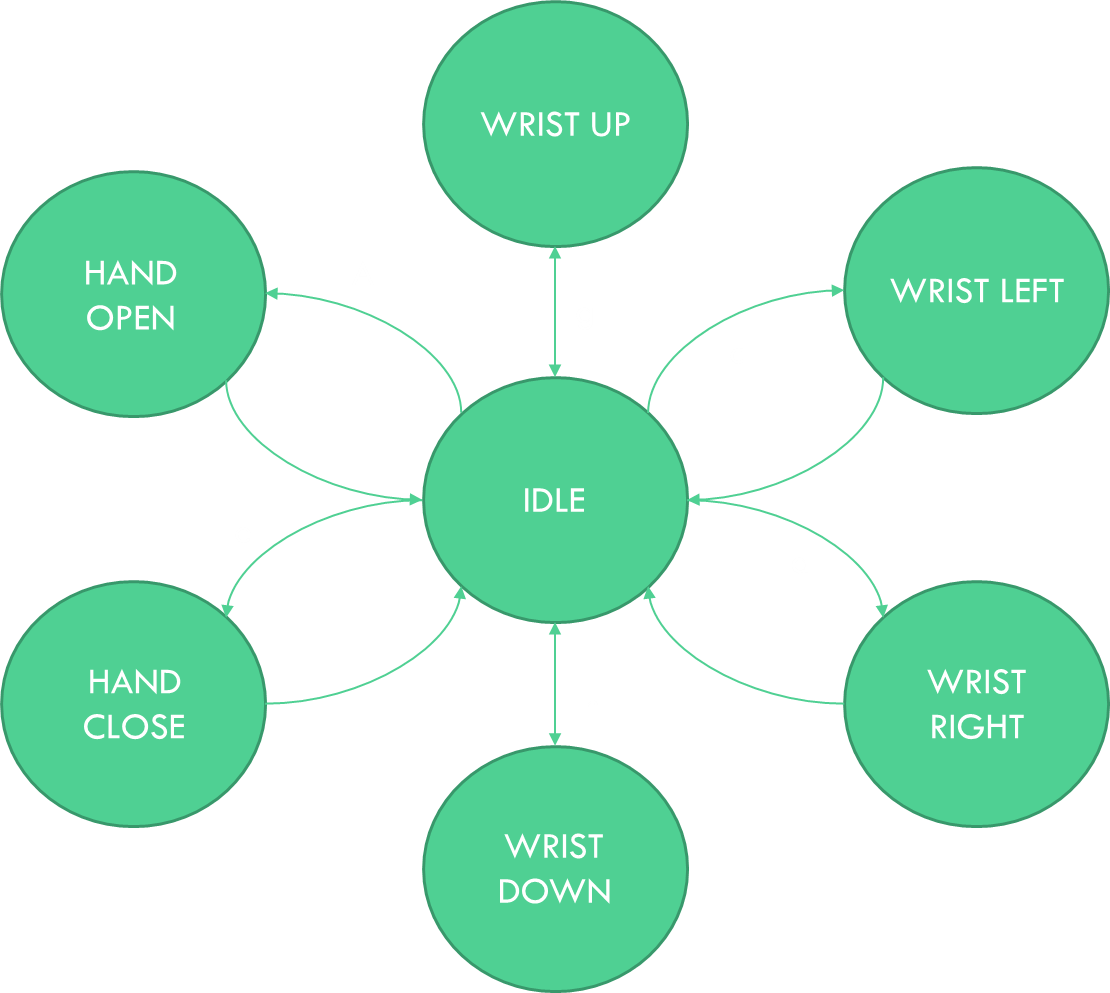


Figure 10: Hand state transition diagram

|  |  |  |
| --- | --- | --- |
| **Action** | **Gesture** | **Associated Key** |
| Switch to Arm Mode | Left hand to the right, quickly | w |
| Switch to Motor Mode | Left hand to the left, quickly | s |
| Wrist Left | Right hand to the left | c |
| Wrist Right | Right hand to the right | d |
| Wrist Up | Right hand up | g |
| Wrist Down | Right hand down | h |
| Hand Open | Left hand up | a |
| Hand Close | Left hand down | e |

### All Modes

|  |  |  |
| --- | --- | --- |
| **Action** | **Gesture** | **Associated Key** |
| Finish | Both hands outward, quickly |  |

Usage Notes

1. The Hi-Technic Motor and Servo controllers time out after a period of inactivity. When they time out, they won't respond to commands, and will act like they are not receiving power. Cycle their power to reactivate them.
2. Ensure the battery being used has sufficient power, otherwise the robot will not function well.
3. You may need a Bluetooth dongle with Cambridge Silicon Radio chipset. See Lego's Bluetooth documentation at <http://www.lego.com/en-us/mindstorms/support/bluetoothsupport/>

# Problems Encountered and their Solutions

When we started working with Bohr, some problems with the servos were pretty apparent. The signals to the arm and hand seemed to have interference issues, and the servos in the hand were continuous-motion where 180-degree servos would be more appropriate. We discovered that the shoulder horizontal motor was the cause of the interference, and swapped it out for a different motor, which solved the problem. We then replaced the three servos in the hand with 180-degree servos. The arm behaves much better now than it did when we started. The programming language used for the NXT codebase does not handle continuous-motion servos to our satisfaction, so replacing those servos greatly improved hand control.

# Translating this Knowledge to ECE 479/579

Next term, we will build on our knowledge from this quarter to add more fine-grained control to Nixon-Bohr, something we didn't have time to do for this quarter. We would especially focus on the arm in the next quarter, since the wheels work well, and the arm is much more technically challenging than the motors. Currently, motors and servos have 3 states: back, forward, and neutral. Adding more states would greatly improve control of Nixon-Bohr. Solidifying the code to control the motors allows us to start from a good baseline next quarter. We can also work more on the head, since we don't have any head control at the moment. We could add a camera on the head and allow the head's orientation to be changed to “look around.” This would allow Nixon-Bohr to be more autonomous than he is now, able to travel corridors to about 30 feet away.

# Technical or Research Problems

1. Nixon-Bohr is heavier than his previous incarnation was. Adding more power to his wheels would help a lot on this. One way to do this would be to add motors to the back wheels. These motors could be in series with the motors on the front, or they could be independent.
2. Nixon-Bohr's face is not utilized by our code. Later versions of our code could be used to mimic the facial expressions of the controller.
3. The arm of the robot is mounted to the base of the robot. This arm could be moved to where the arm should be, mounted to Nixon-Bohr's shoulder, so he could shake hands.
4. Nixon-Bohr has no sensors; he is entirely remote-controlled. Adding bumpers, range-finders, and/or cameras in useful locations could significantly improve the operation of the robot. Adding a camera around Nixon-Bohr's eyes could send feedback to the controller's computer, adding a first-person view of what the robot is seeing.
5. Using the newer version of the Kinect could add more fine-grained control to the robot. The newer version of the Kinect adds finger tracking, meaning the controller's arm and hand could be used directly to control Nixon-Bohr's arms and hands.
6. Add larger wheels to allow Nixon-Bohr to navigate more complex terrain than carpets.
7. Add control for a second arm
8. Add an on-off switch to the battery. Right now the cable to the battery is being disconnected to turn the robot off
9. The DC motor controlling the vertical movement of the shoulder could be replaced by a stepper motor to improve shoulder functionality.
10. Use a more responsive communication protocol than Bluetooth. Many robots have their own protocol that runs on the 2.4 GHz band; one of these would have a better response time than Bluetooth at the cost of implementation simplicity.

# Instructions

**Step by Step:**

1. Setup FAAST and Kinect and interface between the two
   1. To start download and install the Microsoft Kinect for windows SDK. It doesn't matter if you have a XBOX 360 Kinect; it will still work. [Click Here to download](http://www.microsoft.com/en-us/kinectforwindows/develop/)
   2. Download FAAST. FAAST translates movements on the Kinect into functions on a keyboard or keypad. [Click Here to download](http://projects.ict.usc.edu/mxr/faast/)
   3. Connect your Kinect to your PC and allow the drivers to install from the SDK and the Windows Database.
   4. Once the drivers have been installed, open the FAAST program that you downloaded earlier.
   5. From the Tracker drop down menu, select Microsoft to use the Kinect SDK's skeleton framework. This comes with the Kinect with the drivers you've installed.
   6. Click the connect button.
   7. Next, click Gestures. Here you can map your movements to key presses. We have provided a profile in our source code to use.
2. Click [here](http://www.extremenxt.com/vbpart1.htm) for instructions on setting up a Visual Basic script to communicate over Bluetooth between the computer and the Lego NXT brick.
   1. Notes: We will be modifying the Visual Basic script so that the first textbox will be for debug purposes and the second textbox will be for operational purposes.
   2. Additional instructions not on the website:
      1. Add a second textbox and a third label.
      2. Make the third label say “Auto Send”
      3. Make the second textbox say “button click” when the textbox changes.
      4. Add code to clear the text box at the end of the function.
3. Install ROBOTC on your computer and setup the IDE for your application (connect NXT to PC through USB)
   1. ROBOTC costs money, but has a 30-day free trial
   2. Use the configuration wizard to setup servos and motors
      1. Go to menu: ROBOTPLATFORM TYPELEGO NXT + TETRIX/MATRIX
      2. Go to configuration wizard: ROBOTMOTORS AND SENSORS SETUP
         1. Click tab: \*\*\*
            1. X
         2. Click tab: \*\*\*
            1. X
         3. Click tab: \*\*\*
            1. X
4. Copy the ROBOTC source code, referenced below, to the IDE. Compile it. The code should compile with no errors.
5. Download the ROBOTC source code to the NXT.
6. Connect the computer to the NXT via Bluetooth connection using the Visual Basic code. You should hear a beep when the Bluetooth connects.
7. Start the program. If a Bluetooth connection has not been established, it will beep loudly and halt the program.
8. Start FAAST. Select “Start Emulation.” Select the textbox in the VB application
9. Move back until FAAST sees your skeleton. You may now control the robot. The supported motions are documented above, or you can create your own.

# Project Code

## Visual Basic Code:

## FAAST Profile:

## ROBOTC Code:

### 