DESIGNING A CONTROLLER FOR LEGO ROBOT USING C++ LANGUAGE

Mohammad Fotouhi, Ali Eydgahi, and Daniel A. Herz
University of Maryland Eastern Shore
Princess Anne, MD 21853

Abstract

This paper presents the details of an undergraduate senior design project in our design technology course. The main objective of the course is to give students some design experience using the knowledge they have gained in the technology program at the University of Maryland Eastern Shore to produce something that is both innovative and creative. The LEGO Mindstorm robotics system has the capability to be a powerful robot design tool, but not with the provided software called ROBOLAB. The LEGO Mindstorm has a RCX, which is a large LEGO brick hosting a battery case, a Hitachi H8 microcontroller, an IR transceiver, a simple LCD panel, a few control buttons, sensor and actuator connectors shaped into the form of LEGO brick connectors, and some auxiliary circuitry. This project utilizes the power of the RCX by using C++ as the programming language to create a surveillance robot. This allows more programming options and a higher degree of control of the robot. Key goals are motor control, sensor data acquisition, and alarm status. Sample programs are provided to demonstrate the language effectiveness. A surveillance application using touch sensors, motor controls, and a rotation sensor is also presented.

Introduction

The Mindstorm is an interesting system[1-3] that can be used to introduce students to the world of microprocessor controlled robotics. The base system consists of the RCX, an infrared transceiver, and a PC. Additional components, such as motors, sensors, and other building elements can be combined with the base system to allow the creation of functional autonomous robotic devices.

Many robots come with software that are specifically made for the programming of that particular robot. However, these software have limited functions and the use of a more powerful programming language allows the designers to have better control and more options in their design. The LEGO Mindstorm System is no different. It has the capability to be a powerful robot design tool, but not with the provided software, ROBOLAB. To overcome these limitations, the C++ programming language[4-6] which is a more powerful language than ROBOLAB software was selected. Using the three inputs and three outputs of the RCX, the user is able to increase the power of the RCX through software to better motor control, to get better sensor data acquisition, and to use more than one sensor per input.

The objective of this project was to create a more powerful robot system using C++ programming language so that more options in the design process are available and the potential of the RCX command center is fully realized. To demonstrate the usefulness of these options, a robot was designed using LEGO Mindstorm kit and was programmed to follow several paths in its attempt to surveillance an area and to sound an alarm in case it recognizes an intruder. In this paper, we present the results of our involvement and attempts at programming the RCX using C++. We discuss programming the robot using several different programs which perform different tasks. The first will be a surveillance application using touch sensors, motor controls, and a rotation sensor. The second will involve more motor control, turning the robot 180 degrees, while in surveillance mode. Finally, a program to control the movement of robots through use of a light sensor.

The Design Project

The following Materials and Equipment were used in this project:

- One LEGO Mindstorm robot system,
- One RCX command center,
- One infrared download interface,
- Two motors,
- Two push button sensors,
- One rotation sensor,
- One light sensor,
- An IBM compatible computer with GNU C capabilities.

The robot was assembled using the LEGO Mindstorm robot kit. Almost everything one needs can be found in the kit. We decided on a simple roverbot design as shown in Figure 1, but the design can be whatever one wants. The LEGO system allows for a lot of creativity and one is only limited by his/her imagination.

The user decides on what sensors should be used and how to incorporate them into his/her design. In our design, the touch sensors were placed on the ends of front and back bumpers. The rotation sensor was connected to the motors by way of a sprocket so that it could sense motion of the wheel as it travels. Also, the light sensor was placed at the front of the robot facing down so it could recognize the ground changes better. All sensor placements and alignments were considered before the robot was built. It is very helpful to have an initial plan since it will save a lot of time.

One important point to consider in the initial design is the location or position of the RCX command center, which consists of both the power supply and the brain of the robot. If the robot is going to be programmed at all, it will need the RCX as part of its design. Also, the front of the RCX contains the infrared download transceiver and should not be blocked or it will not be able to download the program using the infrared interface.

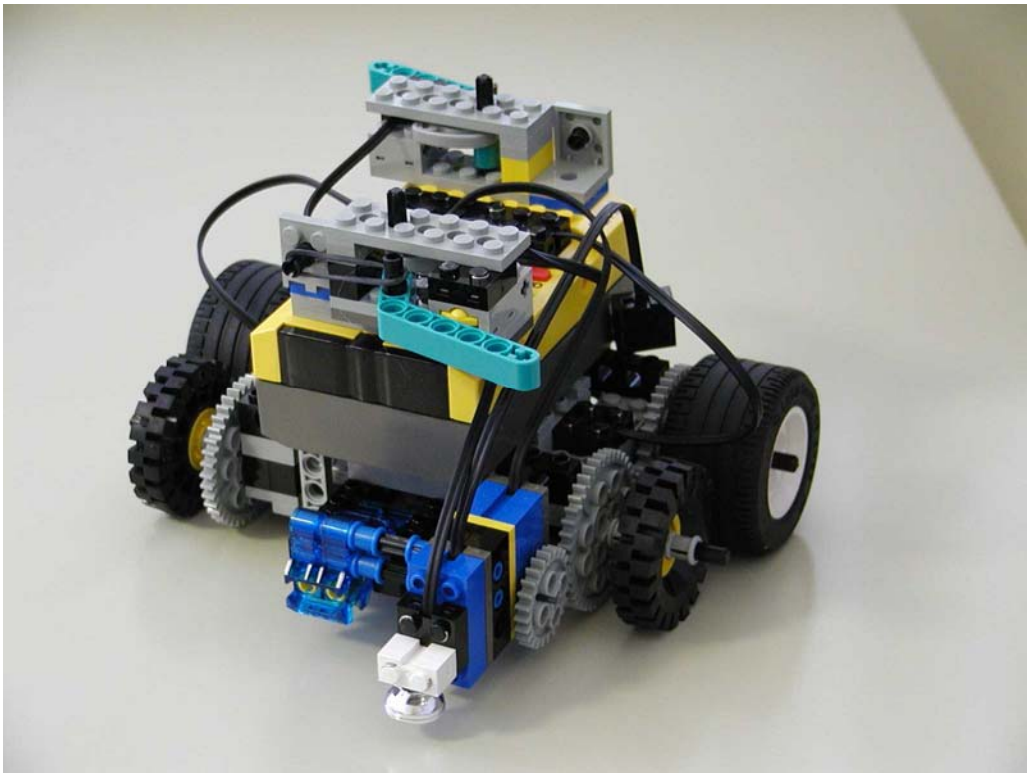

Figure 1: Roverbot .

Infrared Interface and RCX

A LEGO Mindstorm Robot consist of a programmable LEGO brick, called the RCX, which contains three sensor inputs, three actuator outputs, four user buttons, a simple LCD display, an IR transceiver, and a Hitachi H8 microcontroller with 32 kilobytes of RAM, where 4 kilobytes of which is used for interrupt vectors and other low level data.

At the core of the RCX is a Hitachi H8 microcontroller with 32K of external RAM. The microcontroller is used to control three motors, three sensors, and an infrared serial communications port. An on-chip, 16K ROM contains a driver that is run when the RCX is first powered up. The on-chip driver is extended by downloading 16K of firmware to the RCX. Both the driver and firmware accept and execute commands from the PC through the IR communications port. Additionally, user programs are downloaded to the RCX as byte code and are stored in a 6K region of memory. When instructed to do so, the firmware interprets and executes the byte code of these programs. The RCX is shown in Figure 2 and its specification is given in Table 1.

Table 1: RCX specifications.

Series	H8/3297
Product name	H8/3292
Part number	HD6433292
ROM size	16K
RAM size	512
Speed	16MHz @ 5V
8-bit Timers	2
16-bit Timers	1
A/D Conversion	8 8-bit
I/O pins	43
Input only pins	8
Serial port	1
10mA outputs	10

Figure 2: RCX.

The infrared interface consists of the infrared transceiver circuit and the serial cable. The cable to connect the IR transceiver to PC is a null modem cable that contains six wires, of which only five are used. Pin 4 connects to pin 4 but is unused; pins 1, 6, and 9 have no connection. Table 2 shows the pin allocations of the cable.

Table 2: Pin allocations of the cable.

Pin	To	Name	Description
2	3	RD	Receive Data
3	2	TD	Transmit Data
5	5	SG	Signal Ground
7	8	RTS	Ready To Send
8	7	CTS	Clear To Send

The RTS/CTS signals are not used for flow control. Instead, they are used by the PC to check whether or not the transceiver is connected. The transceiver wires CTS and RTS together; the PC checks for the transceiver by asserting and deasserting RTS. If it sees that CTS tracks RTS, then it assumes that the device sitting on the serial port is the transceiver.

Software Development

Since the two programming environments provided by the LEGO group are severely limited in their ability to fully utilize the computational power of the RCX, a number of independent programming environments have been developed for the RCX, including the GNU C Compiler, which is part of the GNU Compiler Collection, to compile C source code to the machine code of the Hitachi H8 processor. The user needs to use this programming environment or one compatible to it in order to use C++ as a coding language for the RCX.

Once the C++ program is written and compiled, the following steps are taken to download and run the program on the RCX:

1. The IR transceiver cable as is specified in table 2 is attached to the serial port of the computer. If computer uses a 25-pin male serial input, it is required to use a 9-pin male to 25-pin female adapter.
2. Face the transceiver towards the front of the RCX, which contains an infrared download transceiver as is shown in figure 3. Allow 4-6 inches between the RCX and the transceiver.
3. Now, the RCX is ready to receive the programs. The program environment provides an option to download the program to the RCX. Each program is downloaded and receives a program number in the RCX.
4. To run the first program, one should push the ON/OFF button on the RCX to turn the RCX on. Then, the PRGM button is pushed until the number 1 is displayed in the LCD window. Finally, RUN button is pushed.

The RCX will execute the first program then will stop. The other programs can be accessed in the same way by going to the next program number and pushing RUN button on the RCX. Each program is assigned a number depending on its download position, i.e. program 1 is downloaded first, program 2 is downloaded second, and so forth. Up to five programs can be downloaded to the RCX.

There has been work done in the area of software development to make the LEGO robot system a more powerful design package. New sensor development has also enhanced the original Mindstorm package. Since the standard programming environments provided by LEGO allow only a very limited access to the resources of the microcontroller, a new environment was needed to enable full access to the power of the RCX. When programming at the machine language level (either using an assembler or through a high level language), a programmer has direct access to both the hardware and to the ROM routines. Hitachi H8 uses memory mapped I/O, and the actual hardware ports are mapped to the highest part of memory, at a so-called eight bit area. The I/O port map, as used by the RCX, is partially illustrated in Table 3.

Table 3: Map of the I/O port.

Memory Range	Function
F000 – F0FF	Motor Control
FFB7	IR Transceiver Range, Button Input
FFBA	IR Control, External RAM Power Save Mode
FFBB	Sensor Power, Timer, LCD I/O
FFBE	Sensor Input, Button Input
FFC3	Serial/Timer Control
FFE0 – FFE4	Sensor A/D Input

The ROM contains a large number of routines, too many to list here but some of the routines and their functionality are listed below.

- Initialization functions and a simple main loop
- Default interrupt handlers
- Memory move, copy, clear, etc. auxiliary functions
- Battery power management

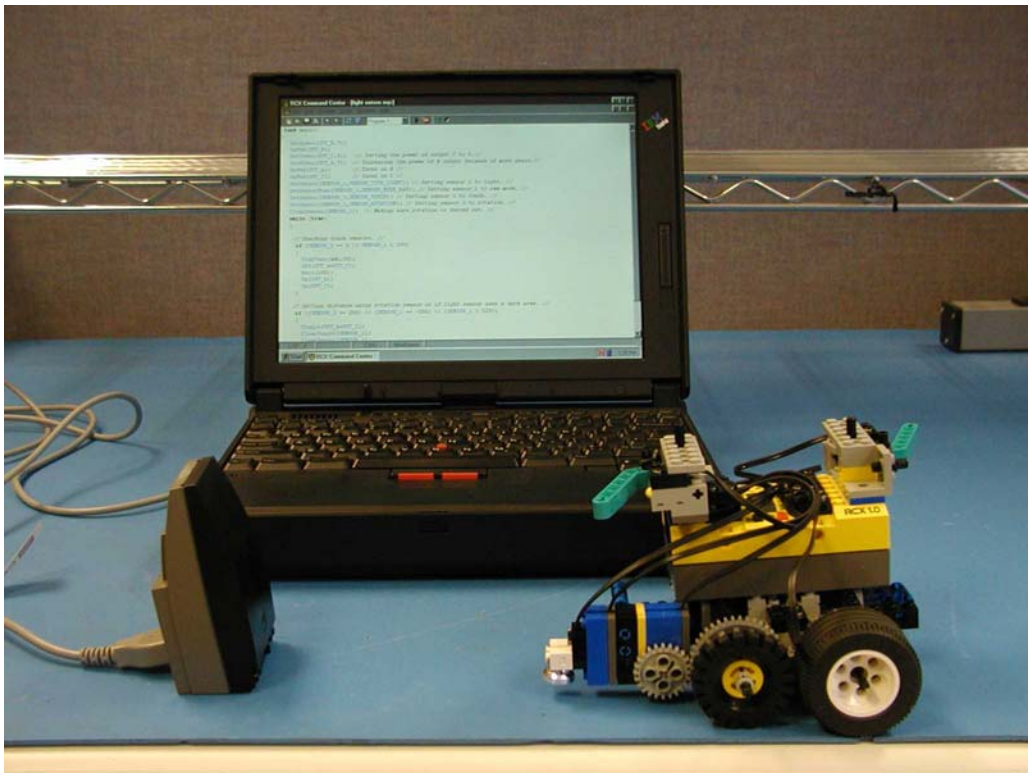

Figure 3: Complete set up.

- Sensor I/O taking care of interpreting raw data
- Motor control
- LCD and sound output
- IR transceiver I/O

The firmware code can be loaded from internal storage through IR and is stored in a memory area starting at hex 8000. The default firmware is meant to interpret user programs expressed in the form of byte code. Figure 3 shows the complete set up of the project.

Programming Examples

The motor control using ROBOLAB program is limited to motors being either full on or full off, and either reverse or forward. Whereas, the C++ programming language can be used to develop a wide range of motor control functions. For example, programs can be written to adjust the motor's direction and power, toggle the motor, or place the motor in a "Float" state, which allows the motor to idle as if it were in neutral.

The C++ program for motor control determines how motor will be controlled. Once the developed motor control program is downloaded, the motors are controlled by the actuator outputs A, B, and C on the RCX. Within the program, there should be motor control statements that identify the specific actuator to be engaged. It should also direct the RCX as what to do with the identified output. For example, OnRev(OUT_A) statement, directs the RCX to turn on actuator A in the reverse direction.

The following statements are example of codes that are related to motor control:

- *On('motors')* Switches the motors on
- *Off('motors')* Switches the motors off
- *Float('motors')* Switches the motors to idle
- *Fwd('motors')* Switches the motors forward (but does not make them drive)
- *Rev('motors')* Switches the motors backwards (but does not make them drive)

- *Toggle('motors')* Toggles the direction of the motors (forward to backward and back)
- *OnFwd('motors')* Switches the motors forward and turns them on
- *OnRev('motors')* Switches the motors backward and turns them on
- *OnFor('motors','ticks')* Switches the motors on for ticks' time
- *SetOutput('motors','mode')* Sets the output mode (OUT_ON, OUT_OFF or OUT_FLOAT)
- *SetDirection('motors','dir')* Sets the output direction (OUT_FWD, OUT_REV or OUT_TOGGLE)
- *SetPower('motors','power')* Sets the output power (0-9)

The LEGO Mindstorm robotic system comes with a variety of sensors that can be integrated into the design so that the robot can interact with the world around it. These sensors can be made even more sensitive and useful when they are controlled with C++. There are many advantages to using C++ when setting up and reading the sensors. One advantage is being able to set up each input to what ever type and mode of sensor the user would like to have. One mode that we talk more about is the "SENSOR_MODE_RAW" mode. In this mode, the value, when checking the sensor, is a number between 0 and 1023. It is the raw value produced by the sensor. What it means depends on the actual sensor. For example, for a touch sensor, when the sensor is not pushed the value is close to 1023. When it is fully pushed, it is close to 50. When it is pushed partially the value ranges between 50 and 1000. So, if the user sets a touch sensor to raw mode, we can actually find out whether it is touched partially. When the sensor is a light sensor, the value ranges from about 300 (very light) to 800 (very dark). This gives a more precise value than using the SetSensor() command. Another advantage of the raw mode is allowing the user to put more than one sensor on a single input. We can connect a touch sensor and a light sensor to one input. Set the type to light (otherwise the light sensor won't work). Set the mode to raw. In this case, when the touch sensor is pushed, we get a raw value below 100. If it is not pushed, we get the value of the light sensor which is never below 100.

The following programs use this idea. The robot must be equipped with a light sensor pointing down, and a bumper at the front connected to a touch sensor. Both of them should be connected to input 1. The robot will drive around randomly within a light area. When the light sensor sees a dark line (raw value > 750) it goes back a bit. When the touch sensor touches something (raw value below 100) it does the same.

Program 1:

This program starts the robot to make random movements until one of the sensors is tripped, then the robot stops back up and continues it random movements. Note that both sensors are connected to input 1.

```
int ttt,tt2;
task moverandom()
{
  while (true)
  {
   ttt = Random(50) + 40;
   tt2 = Random(1);
   if (tt2 > 0)
    { OnRev(OUT_A); OnFwd(OUT_C); Wait(ttt); }
   else
    { OnRev(OUT_C); OnFwd(OUT_A);Wait(ttt); }
   ttt = Random(150) + 50;
   OnFwd(OUT_A+OUT_C);Wait(ttt);
   }
}

task main()
{
  start moverandom;

SetSensorType(SENSOR_1,SENSOR_TYPE_LIGHT);

SetSensorMode(SENSOR_1,SENSOR_MODE_RAW);
     while (true)
     {
      if ((SENSOR_1 < 100) || (SENSOR_1 > 750))
```

```
    {
      stop moverandom;
      OnRev(OUT_A+OUT_C);Wait(30);
      start moverandom;
    }
   }
  }
```

Program 2:

This program turns on and set the power of the two motors. It monitors the rotation sensor and when a specific number of revolutions have completed, the robot will reverse direction. At all times, the robot is also monitoring its touch sensors to warn of an intruder.

```
Task main()
{
  SetPower(OUT_C,5);   // Setting the power of
       output C to 5.//
  SetPower(OUT_A,7);  // Increasing the power of
      A output (more gears).//
  OnFwd(OUT_A);       // Turns on A //
  OnFwd(OUT_C);       // Turns on C //
  SetSensor(SENSOR_1,SENSOR_TOUCH);
     // Setting sensor 1 to touch. //
  SetSensor(SENSOR_3,SENSOR_TOUCH);
     // Setting sensor 3 to touch. //
  SetSensor(SENSOR_2,SENSOR_ROTATION);
     // Setting sensor 2 to rotation. //
  ClearSensor(SENSOR_2);  // Making sure
    rotation is zeroed out. //
  while (true)
  {

   // Checking touch sensors. //
    if (SENSOR_3 == 1 || SENSOR_1 == 1)
    {
      PlayTone(440,50);
      Off(OUT_A+OUT_C);
      Wait(100);
      On(OUT_A);
      On(OUT_C);
    }

   // Setting distance using rotation sensor. //
    if ((SENSOR_2 == 200) || (SENSOR_2 == -200))
    {
       Toggle(OUT_A+OUT_C);
      ClearSensor(SENSOR_2);
     }
    }
  }
```

Program 3:

This program performs the same task as program 2 with one difference. It does a little more with motor control, by turning the robot around, after a specified number of revolutions of the rotation sensor.

```
task main()
{
  SetPower(OUT_C,5);   // Setting the power of
      output C to 5.//
  SetPower(OUT_A,7);  // Increasing the power of A
     output(more gears).//
  OnFwd(OUT_A);       // Turns on A //
  OnFwd(OUT_C);       // Turns on C //
  SetSensor(SENSOR_1,SENSOR_TOUCH);
     // Setting sensor 1 to touch. //
  SetSensor(SENSOR_3,SENSOR_TOUCH);
    // Setting sensor 3 to touch. //
  SetSensor(SENSOR_2,SENSOR_ROTATION);
    // Setting sensor 2 to rotation. //
  ClearSensor(SENSOR_2);  // Making sure rotation
    is zeroed out. //
  while (true)
  {

   // Checking touch sensors. //
    if (SENSOR_3 == 1 || SENSOR_1 == 1)
    {
      PlayTone(440,50);
      Off(OUT_A+OUT_C);
      Wait(100);
      On(OUT_A);
      On(OUT_C);
    }

   // Setting distance using rotation sensor. Robot
      turns 180 degrees. //
    if ((SENSOR_2 == 200) || (SENSOR_2 == -200))
    {
     OnRev(OUT_C);
     Wait(200);
     OnFwd(OUT_C);
     ClearSensor(SENSOR_2);
    }
```

```
  }
}
```

Program 4:

This program uses both the rotation sensor and a light sensor to determine its range of motion. If the robot sees a dark spot on the floor it will immediately change directions and head in the other way. If it doesn't see a change in the floor, then it will continue until the rotation sensor reaches a value of 200. The important aspect of this program is that it demonstrates how two sensors can be connected to a single input.

```
task main()
{
  SetPower(OUT_B,7);
  OnFwd(OUT_B);
  SetPower(OUT_C,5);   // Setting the power of
     output C to 5.//
  SetPower(OUT_A,7);  // Increasing the power of A
     output (more gears).//
  OnFwd(OUT_A);       // Turns on A //
  OnFwd(OUT_C);       // Turns on C //
  SetSensor(SENSOR_1,SENSOR_TYPE_LIGHT);
     // Setting sensor 1 to light. //

SetSensorMode(SENSOR_1,SENSOR_MODE_RA
W); // Setting sensor 1 to raw mode. //
  SetSensor(SENSOR_3,SENSOR_TOUCH);
    // Setting sensor 3 to touch. //
  SetSensor(SENSOR_2,SENSOR_ROTATION);
   // Setting sensor 2 to rotation. //
  ClearSensor(SENSOR_2);  // Making sure rotation
    is zeroed out. //
  while (true)
  {

   // Checking touch sensors. //
    if (SENSOR_3 == 1 || SENSOR_1 < 100)
    {
      PlayTone(440,50);
      Off(OUT_A+OUT_C);
      Wait(100);
      On(OUT_A);
      On(OUT_C);
    }
   // Setting distance using rotation sensor or if light
     sensor sees a dark area. //
    if ((SENSOR_2 == 200) || (SENSOR_2 == -200)
     || (SENSOR_1 > 524))
    {
      Toggle(OUT_A+OUT_C);
      ClearSensor(SENSOR_2);
      ClearSensor(SENSOR_1);
      Wait(100);
    }

  }
}
```

Conclusion

The world of robotics is an exciting and interdisciplinary field which requires mechanical, electrical, and programming knowledge. The LEGO Mindstorm system is a very good choice to introduce students to this field of engineering. However, the LEGO system is not very powerful because it is limited by its ROBOLAB software. By using C++ software, our student developed a multitude of design options and enhanced the overall system. Through this project and with experience in trial and error, the student was able to build better and more powerful robot. The knowledge he gained through this project will help him with future robot developments.

References

1. "LEGO Home" LEGO Mindstorms, LEGO Group, April 2000. http://mindstorms.lego.com/

2. "LEGO Mindstorms", LEGO Products, LEGO Group, April 1997. http://www.mooreed.com.au/competition/datalogging/LEGO%20sensor%20specification.htm

3. "RCX Internals." RCX Internals, KeKoa Proudfoot, April 2002 http://graphics.stanford.edu/~kekoa/rcx/#Hardware

4. Aitken and Jones. Teach Yourself C in 21 Days. Prentice Hall, NJ, 1993.

5. Dale, Nell, Chip Weems and Mark Headington. Programming and Problem Solving with C++. Jones and Bartlett Publishers, Boston, 1997.

6. Nyhoff, Larry. C++ An Introduction to Data Structures. Prentice Hall, NJ, 1999.

Biographical Information

Dr. Mohammad Fotouhi is a Professor of electrical engineering technology at University of Maryland Eastern Shore. He received his Ph.D. in power System Engineering from University of Missouri-Rolla, M.S. from Oklahoma State University and B.S. from Tehran Polytechnic College. He has been conducting practical research on the growth and characterization of the dilute magnetic semiconductor since 1985. He is a member of Eta Kappa Nu Honor Society. He was chairman of Student and Industry Relation and Host Committee member of IEEE Conference on Power Systems Computer Application in 1991. He also was chairman of Student Relation and Host Committee member of the IEEE Power Society Winter Meeting in 1996.

Dr. Ali Eydgahi is a Professor and Chair of Engineering and Aviation Sciences Department at University of Maryland Eastern Shore. He received his Ph.D. and M.S. in Electrical and Computer Engineering from Wayne State University. Since 1986 and prior to joining University of Maryland Eastern Shore he has been with the State University of New York, University of Tehran, Wayne County Community College, and Oakland University. Dr. Eydgahi is recipient of the Dow Outstanding Young Faculty Award from American Society for Engineering Education in 1990, and the Silver Medal for outstanding contribution from International Conference on Automation in 1995. He is the ASEE Campus Representative at UMES and has served as a regional and chapter chairman of IEEE and SME in New York. He also has served as a session chair and a member of scientific and international committees for many international conferences. Dr. Eydgahi has published more than ninety papers in refereed international and national journals and conference proceedings.

Mr. Daniel Herz received his B.S. degree in Electrical Engineering Technology from University of Maryland Eastern Shore in 2002. He started his career as an Engineering Specialist in the Telecommunication branch at the National Security Agency in 1987. After working for five years building and troubleshooting telecommunication systems at NSA, he started working on encrypting devices in the Y42 branch for the final four years of his employment at the agency. In 1996, he went to work as a RF Engineer at Filtronic Comtek. His current interest is to design Microwave filter systems for the cellular phone industry.