SCRAP

Speech Controlled Robot Application

Developed by Matthias Heie
Supervised by dr. drs. Leon Rothkrantz

1 Introduction

This report is the documentation of a project done by Matthias Heie under the supervision of dr. drs. Leon Rothkrantz. The project was done at Knowledge Based Systems Group at Faculty for Information Technology and Systems, Technical University Delft from September 2001 to January 2002.

1.1 Background

Imagine a phone that can simultaneously translate everything spoken into it. When you call your Chinese business partner, there is no need to speak a third language just because none of you are able to speak each other's language. Perhaps you don't even know a third language in common. You just grab your phone, choose Translate on the menu, choose Mandarin (or maybe Cantonese?) as language and dial the number. When your business partner picks up the phone, you can speak to him in your own language and you are answered in the same language. Maybe the voice will be a monotone, clearly synthetic voice. Or perhaps the technology is good enough to produce your business partner's voice, speaking in a natural way?

This little science fiction story shows three aspects of *language technology*:

- automatic speech recognition (ASR), which recognize what words are said.
- machine translation, which translates the words from one language to another.
- text-to-speech synthesis (TTS), which synthetically speaks the translated words.

ASR and TTS are two parts of the sub field *speech technology.* Speech technology is distinguished from other parts of language technology by the fact that it is concerned with language in form of sound signals.

Speech technology has until now not been broadly used in applications. But this is slowly starting to change. For instance train companies are starting to use this technology on automatic answering machines to inform customers about schedules. You are asked to speak out the name of your starting point and destination.
For handicapped people speech technology can mean a revolution. It can be used to control the TV with voice, automatic reading of a newspaper or giving a voiceless person a synthetic voice.

In this assignment the controlling of a robot by speech will be studied. Commands are spoken into a microphone, followed by an action by the robot.

1.2 Problem description

The goal of this project is to make a system with speech interface for controlling a Lego Mindstorms Robotics Invention System (LMRIS) robot using Microsoft Speech SDK.

LMRIS contains building units, hardware and software to build a robot and control it from a PC. By making a system that integrates this software with the software in MS Speech SDK, it should be possible to control the robot with speech.

1.3 Problem strategy

1. The software is installed and studied.
2. A small prototype with very little functionality is made.

This prototype is then further developed in an evolutionary way:

3. A new sub goal is specified as new or improved services.
4. These new or improved services are implemented in the system.
5. Step 3 and 4 are repeated until the quantity and quality of services are considered sufficient.
6. Finally the system is tested and evaluated.

The end product is the system itself with the functionality described in chapter 4.

2 Hardware

The speech is received by a microphone and processed on a PC. When a command for the robot is recognized, the PC sends a command message to the robot's built-in-computer, using infrared signals. The robot's computer analyzes the message and takes appropriate actions.

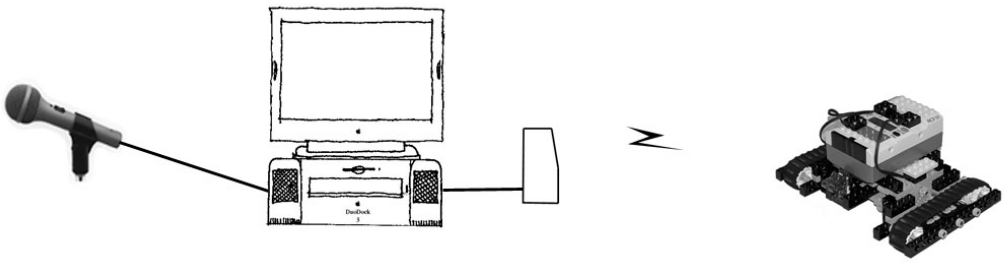

Figure 2.1

2.1 Microphone

The speech of the user is received by a microphone.
During the development and testing of this project a Maxi SM-20 microphone was used. This is not a high quality microphone, and "in general, the microphone should be a high quality device with noise filters built in. The speech recognition rate is directly related to the quality of the input. The recognition rate will be significantly lower or perhaps even unacceptable with a poor microphone." [Microsoft Speech's homepage]
Therefore it is reason to believe that the performance of this system will increase with a microphone with higher quality.

2.2 PC

The speech signals are processed on a PC.
The computer needs to have an available COM port for the infrared sender. Apart from that it needs to satisfy the requirements for the software running on the machine.
The recommended requirements for Microsoft Speech SDK are a Pentium II/Pentium II-equivalent or later processor at 233 MHz with 128 MB of RAM.
The requirements for the Lego Mindstorms Robotics Invention System 1.5, which Lego Spirit.ocx is a part of, are a Pentium I/Pentium I-equivalent or later processor at 166 MHz with 16 MB of RAM.
The requirements of the main application are not investigated but this module is far less computable demanding than Microsoft Speech SDK.

So the bottleneck of the requirements is the Microsoft Speech SDK, and hardware that fulfills its requirements should also fulfill this system's requirements.

2.3 Robot

The commands are sent to the robot. The robot is built using Lego Mindstorms Robotics Invention System 1.5 (LMRIS). This set contains:

- building units to build robots, including motors and sensors.
- a computer called RCX to control the robot, see underneath.
- software and communication devices for communication between the RCX and a PC.

The robot is a belt band vehicle similar to the robot described at page 25 in Constuctopedia for LMRIS. To build this vehicle: look at the preceding pages in the Constuctopedia.

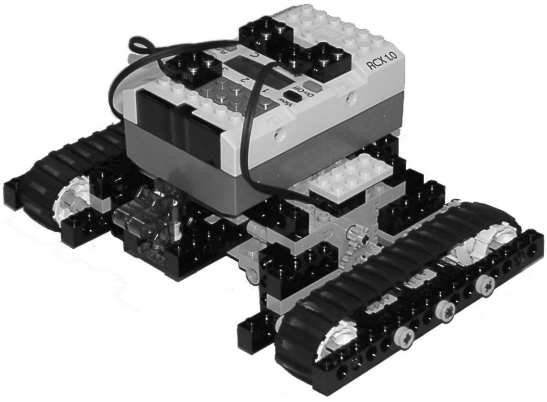

Figure 2.2 Robot

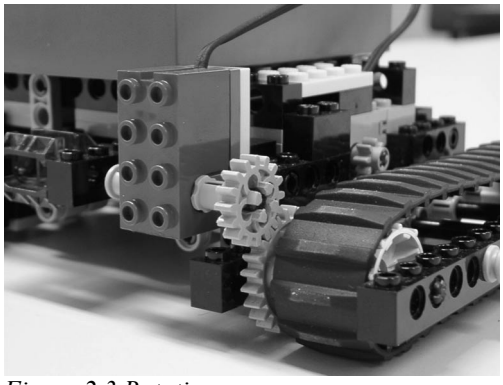

Figure 2.3 Rotation sensor

In addition the robot has a rotating sensor which is used to count the rotations of the wheels. The gear of the sensor is connected to the backmost gear at the right side of the robot (see figure 2.3). The other end of the sensor is connected to input port 3 at the RCX.

The RCX receives messages (immediate commands and programs, see chapter 3.4) from the PC. The immediate commands are executed immediately. The programs (max 5) are stored in its memory and executed when they get an execution command. The computer has three output channels used to control motors and three input channels used to receive sensor information. In this application output channel A and C are connected to respectively the left and the right motor. Input channel 3 is connected to a rotation sensor as described above.

2.4 Communication devices

The PC sends its commands to the robot using infrared signals. The RCX has a built-in transceiver. Another transceiver is connected to a COM port at the PC. At the PC transceiver there is a switch that is used to switch between short term transmission and long term transmission. In this system it should be switched to long term transmission. The best possible transmission conditions are achieved when the transceivers are close to each other, facing each other and no objects are standing between them.

3 Software

The speech received by the microphone is analyzed by a speech recognition engine. The analyzing is done according to a predefined grammar written in a grammar file. The speech recognition engine communicates with a command manager via the speech API. The command manager is a part of the main application and it is notified when there is a recognized phrase. It analyzes the phrase and sends a message to the robot via the Lego Spirit.ocx component, to make it execute appropriate actions.

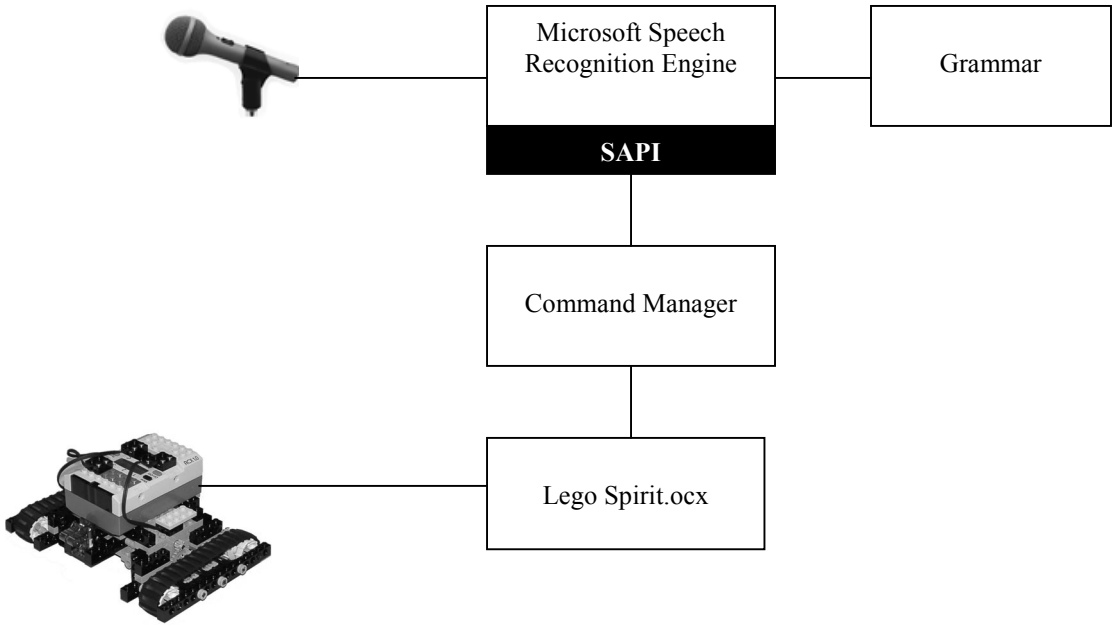

Figure 3.1

3.1 Platform

Supported operating systems for Microsoft Speech SDK, which the Speech Recognition Engine and the Speech API are a part of, are

- Microsoft Windows 2000 Professional, Server, English edition or Windows 2000 Professional with Japanese or Simplified Chinese Language support.
- Microsoft Windows NT® Workstation 4.0, service pack 6a, English, Japanese, or Simplified Chinese edition.
- Microsoft Windows Millennium edition.
- Microsoft Windows 98 (Windows 95 is not supported).

Minimum requirements for Lego Mindstorms Robotic Invention System 1.5, which Lego Spirit.ocx is a part of, are Windows 95/98.
The requirements for the main application, which is compiled to an executable file using Microsoft Visual Basic 6.0, are not investigated, but worked on a Windows 98 machine, where it was developed.

So the platform requirement for the system is Windows 98. It might work on other platforms such as Windows NT, Windows 2000 or Windows Millennium edition, but this is not tested and can therefore not be guaranteed.

3.2 Speech Recognition Engine and Speech API

The goal of this project was to build a speech interface using Microsoft Speech Software Development Kit (MS Speech SDK). This is a software development kit for building speech applications for Windows. In this system version 5.1 of this SDK is used.
MS Speech SDK 5.1 consists of:

- SR (speech recognition) engine files
- TTS (text-to-speech) engine files
- SAPI (Speech Application Programming Interface) files, the API for using SR engine and TTS engine
- example files
- documentation

The TTS engine, which synthesize written text in form of strings or files into spoken audio signals using synthetic voices, is not used in this system.
The SDK can be downloaded for free from Microsoft Speech's homepage, www.microsoft.com/speech, where also the full documentation can be found.

The function of the SR engine is to take a digitalized speech signal as input and convert that into recognized words and phrases. The incoming speech is compared with a predefined grammar.

The SAPI provides a high level interface between an application and speech engines. The SAPI notifies the application when a speech event occurs. A speech event might be for instance the start of a phrase, the end of a phrase or a speech recognition. Microsoft SAPI 5.1 can handle more than 30 kinds of such speech events. In this system our primary interest is speech recognition events. When an event occurs, the application will be notified and it receives a structure with information about the event. In the case of a speech recognition event the structure contains information such as rules activated, the phrase as a string and confidence value of this phrase being uttered. Because our grammar is built up hierarchically, the structure is typically also hierarchically built up.
The application needs a recognizer object to access the SAPI. There are two ways to set up this object:

- Shared resource instance. This set up allows resources such as recognition engines, microphones and output devices to be used by several applications on the same time.
- Non-shared resource instance. This set up allows only one application to control the resources.

The shared resource instance is the preferred option for most desktop applications. In this way more applications can use the microphone. Therefore shared resource instance is used in this system.

Initially the speech recognition uses a default voice profile, which perform fair for every voice. It is possible to configure the speech recognition system to a specific voice. By doing this the performance is supposed to increase for that specific user.

3.5 Grammar

There are two main types of grammar:

- Dictation grammar: In this type of grammar the SR engine will load its own grammar. The grammar will then typically be very comprehensive. Ideally all allowed phrases in a language could then be recognized. This leads to a huge number of possible phrases.
  The advantage of dictation grammar is that (theoretically) all legal phrases can be recognized without having to specify the whole grammar first. The disadvantage is that chances of misrecognition will be very high, at least with today's technology. The term misrecognition is the case when a phrase is being recognized as a different phrase.
- Command-and-Control grammar: In this type of grammar we specify our own grammar for the application. The grammar can then be drastically limited. Only phrases that make sense in our field of application would be included and the chances of misrecognition would be strongly reduced. The disadvantage is that the grammar has to be manually specified.

MS Speech SDK supports both dictation grammar and command-and-control grammar. Because our field of application is quite limited, command-and-control grammar is chosen.

There are two types of command-and-control grammar:

- Static grammar: In this type of grammar the grammar is completely predefined and loaded at execution of the application. It is not possible to change any of the rules during runtime. However, it is possible to achieve a kind of pseudo dynamic grammar by specifying all necessary rules and only consider speech events by the rules that should be active at the moment. But sometimes this is not enough, for instance when the user should be allowed to specify words or names that are not known at development time. It would be possible for the user to change the contents of the grammar file before execution, but this is not an elegant solution. During runtime it may not be possible at all.
- Dynamic grammar: In this type of grammar the contents of the grammar can change during runtime. By allowing this, we solve the problems of static grammar.

MS Speech SDK supports dynamic grammar (and of course also static grammar).
In our system we know at runtime what phrases we want to be recognized, so there is no need for a dynamic grammar. Therefore our system uses static grammar.

The grammar for Microsoft Speech SDK is a context-free grammar written to a file in XML format.

Underneath follows an explanation of the structure of the XML grammar file used in this system. For a complete documentation on the structure of this XML format, see Microsoft's documentation on Microsoft Speech SDK.

```
<GRAMMAR LANGID="409">                  Start of grammar. 409 is the language ID of English

    <DEFINE>                            Definition list, where all properties are
        <ID NAME="Download"             associated with a value.
          VALUE="101"/>                 A property can be a rule or a sub phrase. In this case
                                        the rule Download are associated with the
        ...more definitions...          ID number 101. The ID numbers are used when the
                                        recognized phrase structure is analyzed by the
    </DEFINE>                           command handler.

    <RULE NAME="Download"               Rule
      TOPLEVEL="ACTIVE"….>
        ...rule body...
    </RULE>

    ...more rules...

</GRAMMAR>                              End of grammar
```

As an example of a rule follows a *simplified* version of the Move rule, which is used to make the vehicle drive forward or back

```
<RULE NAME="Move"                       Rule header. Attributes: name of the rule (which is
  TOPLEVEL="ACTIVE">                    associated with a value in the definition list), top
                                        level rule means it can be the start of a phrase (not
                                        only a sub rule).
    <O>                                 A (sub) phrase inside <O>…</O> tags is an optional
                                        part of the main phrase.
        <L>                             <L>…<L> contains a list of alternative (sub) phrases.
            <P>go</P>                   <P>…<P> contains a (sub) phrase.
            <P>drive</P>
            <P>move</P>
        </L>
    </O>
    <RULEREF="Number">                  A pointer to a sub rule. Number is a rule that
                                        represents a maximum three-digit number.
    <P>centimeters</P>
    <L PROPID="Direction">              A list can have a property ID, that will be associated
                                        with a value. This ID and value is used in the
                                        analyzing of the phrase.
        <P VAL="0">forward</P>          The VAL attribute contains the value the property is
        <P VAL="1">back</P>             associated with.
    </L>
</RULE>
```

Examples of valid phrases:

- "go ten centimeters forward"
- "drive fifty-two centimeters back"
- "two hundred and thirty four centimeters forward"

3.3 Command Manager (Main Application)

The command manager/main application is tailor made for this system. It is implemented in Visual Basic. The reasons for choosing Visual Basic are as follows:

- Microsoft SAPI 5.1 supports Visual Basic.
- Lego Robotics Invention System 1.5 contains the Visual Basic component Lego Spirit.ocx. With this component it is possible to send commands and download programs from the PC to the RCX.
- Visual Basic is easy and uncomplicated compared to for instance C++. Yet VB is powerful enough to implement the needed functionality.

The command manager is a part of the main application, which in addition to the command handler's tasks is responsible for the initializing:

1) Creating a recognizer object (shared resource instance).
2) Load the grammar.
3) Initialize the COM port.
4) Initialize the IR sender/receiver.

After this initializing, the main application functions purely as a command manager. Therefore from now on the main application will be referred to as the command manager.

The command manager's tasks are:

1) Listening to the SAPI so it can be notified when a speech event of its interests occurs.
2) Analyzing the properties of the speech event to find the semantic of the event.
3) Carry out appropriate action.

The command manager is listening for two kinds of events:

- False recognition event: an event that occurs when the user probably spoke words (as opposed to f. ex. the sound of coughing) but the SR engine couldn't find a close enough match in its grammar. In this case a message will be written to the main form to inform the user about the false recognition.
- Recognition event: an event that occurs when the user spoke words that is recognized by the SR engine as a phrase in the grammar.

When there is a recognition event the command manager will carry out the following actions:

1) Writing the recognized phrase on the main form.
2) Analyzing the structure, which contains the information of the event. First it will find out what top level rule that was fired to find out what kind of command it was. It might be a download, power, move, turn or stop command.
3) Traversing the structure to find all its properties. By doing this it finds out what slot to download the program to, what direction and distance to move, what angle to turn etc.
4) Sending one or more messages to the RCX (each containing one command) to make it carry out the wanted action, using the Lego Spirit.ocx component. For documentation on this component, see its Technical Reference.

5) Writing on the main form what action was performed.

3.4 RCX-commands and programs

There are two ways of controlling the RCX via the IR transceiver.

- Immediate commands. In this case the RCX receives a command and carries it out immediately. An examples of such a command is *On(MotorList)*, which turns on the motors listed in *MotorList*.
- Programs. More commands can be grouped together to a program. The program can be downloaded to one of the RCX's 5 program slots. One program can consist of several tasks, such that one slot pr task is not needed. Programs and tasks can be executed with immediate commands.

A consideration that has to be made is how much computation should be made on the PC and how much should be made on the RCX. It would be nice to eliminate the need of programs such that there is no need to download any program and maybe delete a program that already exists. However, such a solution is not good. Because a sensor is used to tell when the vehicle has moved the desired distance or has turned the desired angle, the sensor needs to be polled. Doing this on the PC has two disadvantages.

- first a poll message has to be transmitted to the RCX, then the RCX has to read the sensor value and finally the result is transmitted back to the PC. If the value is big enough, the PC has to transmit a message back to the RCX that it should stop. This procedure leads to delays.
- The polling has to be done in a while loop. This while loop takes up a lot of resources and will severely reduce the performance of other tasks on the PC. The RCX has in this system only one task at the time, and is therefore suitable for this polling.

So programs are needed for the more complex operations. Only one program is needed because all tasks can be included in one program.

4 Functionality

The overall functionality of this system is to control a Lego Mindstorms Robotics Invention System robot by speech.

Starting the application

The application will start by executing scrap.exe
A window will pop up. The window contains two lists:

- Phrase Recognition List. Here all recognized phrases will be written. By false recognitions "<No recognition>" will be written.
- Action List. Here all the actions taken by the system as an answer to a command, will be written. Also error messages will be written here.

Quitting the application

The application will be quitted by mouse clicking the cross in the upper left corner of the window.

The rest of the functionality is controlling of the robot. It is done by speaking phrases into the microphone. Every phrase must start with "*my robot*" followed by the command. Ex: "*my robot drive forward*"

Notation:
Sub phrases in square brackets are optional. Ex: "*forward*" has the same effect as "*drive forward*".
Slash denotes alternative sub phrases. The alternative may or may not have the same effects. Ex: "*drive forward*" has the same effect as "*go forward*", while "*drive forward*" has a different effect than "*drive back*".
Parentheses are used for grouping the alternatives.
Angle brackets are used for variables. The variables are of the following two groups:

- {digit}: the phrase representing a digit, "*zero*", "*one*" up to "*nine*"
- {number}: the phrase representing a maximum three digit number, f.ex. "*one hundred and fifty three*" or simply "*three*".

Downloading

To be able to execute the more advanced commands, a program has to be downloaded to one of the RCX's 5 program slots.

Command: "*download* [*program*] [*to slot* [*number*] <slotnr>]"
<slotnr>: {digit} that indicates which program slot the program should be downloaded to. Default: 5
If <slotnr> is 0 or between 6 and 9, the system will only generate an error message.

Moving

The robot can move forward and back, either a specified distance or until another command is given.

Command: "[*go/drive/move*] [<distance>] (*forward/back*)
<distance> can be specified three ways:

- "<meters> (*meter/meters*) [*and* <centimeters>]"
- "<centimeters> *centimeters*"
- "[<meters>] *point* [<decimal>] *meters*"

<meters>: {digit} that indicates number of meters the vehicle should move
<centimeters>: {number} that indicates the number of cm the vehicle should move
<decimal>: {digit} that indicates 10*number of cm the vehicle should move. Be aware that only one decimal is accepted.

Default: infinity, i. e. the vehicle will move until another command is given.
"(*forward/back*)" indicates the direction of movement.

Stopping

The robot can be stopped, i. e. the motors will be turned off.

Command: "*stop*"

Turning

The robot can turn left of right a specified angle. If the command is given while the robot is moving, the robot will stop, rotate the specified angle and then move on again.
If the command is given while the robot is standing still, the robot will rotate the specified angle and then stop again.

Command: "[*go/drive/move/turn*] [<angle> *degrees*] (*left/right*)"
<angle>: {number} that indicates the number of degrees the vehicle should turn. Default: 90
"*left/right*" indicates direction of turning.

Power adjustment

The power level of the motors can be adjusted. The power level is between 0 and 7.

Command: "[*set*] *power* [*to*] [*level*] <power level>"
<power level>: {digit} that indicates power level. If <power level> is 8 or 9 the system will only generate an error message.

Remark

If a command is given while another operation is being executed, the first operation will be interrupted. When the second operation has finished, the first command will not be continued. Example: the robot first receives a command to drive 50 cm forward. Before this operation is finished a command to turn 90° left is given. Then it immediately starts turning left. When it has turned left, it will stand still.
There is one exception: When the robot is moving an infinite distance (given by the commands "*forward*" or "*back*" with no specified distance) and a turning command is given, the robot will continue forward or back after the turning operation is finished.

5 Test

The functionality of the system can be divided into two parts:

- the speech recognition part, which is responsible for recognizing the phrases.
- the robot controlling part, which is responsible for the right reactions to the commands.

5.1 Speech recognition part

First the speech recognition part of the system is tested.

To make a test with quantitative results is very difficult. With speech it is almost impossible to fully control the variables. Every person has a different voice. No one is able to say a phrase in the exact same manner two times. The background noise will vary constantly.
So the test design used is a more qualitative approach:

- Three persons with different quality of spoken English are chosen as test persons.
- The test is done in a room with little background noise.
- After a little practicing, the test person is presented a sheet with typical commands. They vary from the simplest commands ("*stop*", "*back*", etc.) to more complex ones ("*drive one hundred and forty five meters right*" etc.)
- The performance of the system is registered.
- In the end the results are evaluated without the use of statistics

.
The three test persons were:
Test person 1: a man in his mid-20s from Eastern Europe. His English is well but his Slavic accent is obvious. He uses the system for the first time.
Test person 2: a man in his mid-20 from Australia. He is a native speaker of English. He uses the system for the first time.
Test person 3: a man in his mid-20 from Scandinavia. His English is well and the accent is also quite good. He is the developer of this system and is therefore an experienced user.
Remarks: The choice of test persons varies in quality of accent and experience with the system. With a bigger test panel it would have bee possible to consider other variables such as age and sex.

Results for test person 1:
The simplest phrases are recognized quite well. Misrecognition or non-recognition happens, but in most cases the phrases are being recognized.
The more complex phrases are a bigger problem. Misrecognition or non-recognition happens very often. The numbers seems to be a big problem.

Results for test person 2:
In general the results are better for this persons. But still misrecognition and non-recognition happens quite often on the more complex phrases. The numbers are

performing much better, but the biggest problems are teens vs tens (f.ex. 16 vs. 60) and 13/30 vs 15/50.

Results for test person 3:
Here the results are good. Occasionally misrecognition or non-recognition happens. Sometimes irrelevant mistakes are done, such as drive being recognized as move. The problem with teens vs tens is not present anymore, but the problem with 13/30 vs 15/50 is still present.

Evaluation:
The improvement from test person 1 to test person 2 shows that a good English accent is necessary for good results. But the improvement form test person 2 to test person 3 shows that experiences with the system is also important. Why? My theory is the following: Sometimes a native speaker can be quite hard to understand, both for a human and a machine, because of a nonchalant way of speaking. On the other hand the developer of this project (test person 3, which is not a native speaker) got the understanding for how important intonation is. The volume of the voice, the intonation, the stressing of sounds etc were adjusted over months to make the system perform better until the right way of speaking came natural. For instance the problem with teens vs tens is not there anymore because the user has learned that it is important to stress the last syllable in “sixteen”.

This shows that although it is possible to train the system to the user (not done is this testing), it is important to train the user to the system. So if the user has quite a good English accent and he is used to the system, then it can perform well.

5.2 Robot control part

Secondly the robot control part of the system is tested.

To test the robot control part is much easier. If a command is not recognized properly, the command can be spoken again until correctly recognized.
A list was made with representative commands. A wanted result for each command was specified. Then the commands were spoken. In case of false recognition, the commands were spoken again until the right commands were recognized. The results were registered and finally compared to the wanted results.
The results are shown in Appendix B.

Evaluation:
The robot control part of the system works perfectly. This is not surprising, because the VB program and grammar file has a reasonable size, such that every algorithm could be modified until they worked perfectly during the development.

6 Conclusion

The goal of this project was to develop a speech controlled robot application by using Microsoft Speech SDK and Lego Mindstorms Robotics Invention Systems. The goals were fulfilled with quite good results. In this case Visual Basic was used, a programming language that is easy to learn and use but yet powerful enough to make robot control systems. With fairly limited code it is possible to reach high robot control performance. The bottleneck of the performance lies on the speech recognition part of the system. For users with a strong accent the performance will not be very high. For people with a good accent, and even for native speakers, it takes a while to learn how to speak to achieve good results. When this is learned, the speech recognition performance will be quite high, although not perfect. The performance can be increased by training the system to a specific voice (not done in this project) and defining a strict grammar were the developer also have in mind what words can be confused. However, it will not be a good idea to use this speech software in systems were false recognitions will lead to fatal errors, f. ex. in planes. But the security of the system can be increased, for instance by implementing a confirm mechanism.
Still much of the performance results depends on the speech software, which hopefully will improve over the next years.

7 Remarks

7.1 Possible further developments

Implement more functions. The Lego Spirit.ocx component and RCX consists of far more functionality than used here, so the functionality can be extended.

By using lights, camera and sensors such as light sensor or touch sensor, multi modality can be implemented.

Implement functionality that makes use of Text-To-Speech to make a dialogue system. A TTS engine is part of the MS Speech SDK and should be easier to use than the Speech Recognition Engine.

Make systems were more than one robot is used.

Combine this system with existing systems developed on the KBS group, such as Hit-And-Run or Follow-The-Black-Line.

Make use of Internet. For instance by audio streams, such that a robot can be controlled from a different location.

7.2 Possible further improvements

The possibility of training the speech engine to a specific voice was not used. This would probably have increased the performance.

Not all the files in MS Speech SDK are necessary in this system because the TTS engine is not used. It would therefore have been a good idea to make a special installation package for this system.

The accuracy of move and turn operations are not very high. The reason for not putting more effort into this part is that this project was considered to primary a speech technology project.

It would have been possible to use a better microphone. This would probably have led to better results.

7.3 Tips

Some advises when speaking:

- Don’t speak to fast but also not to slowly

- Speak clearly.
- Don’t speak too close to the microphone. 30-50 cm is a good distance.
- Try to speak with a good English accent.

8 Litterature

Microsoft Speech's homepage:
www.microsoft.com/speech
On this site Microsoft Speech SDK can be downloaded for free and its full documentation can be found.

Lego Mindstorms SDK's homepage:
www.legomindstorms.com/sdk
On this site the documentation of Lego Spirit.ocx can be downloaded.

SWAMP (Speech interfacing in the Wireless Automotive Messaging Pilot)
Master's Thesis of Chen-Ke Yang
Extensive documentation on development of a system with speech interface by the use of MS Speech SDK.

Appendix

A. Tutorial

This tutorial describes how to use SCRAP to control a robot built by using Lego Mindstorms Robotics Invention System (LMRIS). For a full description of the functionality, see chapter 4.

Robot building

The robot is illustrated in LMRIS Constructopedia, page 25.

1. If the robot is not yet built, it should be built according to the instructions on the preceding pages in Constructopedia.
2. Check that the left motor is connected to output port A and that the right motor is connected to output port C.
3. Check that the rotation censor is connected to the backmost gear on the right in one end and connected to input port 3 in the other end (see figure 2.3).

Installation

Microsoft Speech SDK and Lego Spirit.ocx has to be installed to use this system. MS Speech SDK can be downloaded for free from Microsoft Speech's homepage: www.microsoft.com/speech. The installation instructions are also found on this site. Lego Spirit.ocx is on the cd-rom in LMRIS. The documentation can be found on www.legomindstorms.com/sdk

Hardware setup

1. Connect a microphone to the microphone port.
2. Connect the IR transceiver to COM port 1.
3. Place the IR transmitter about 1 meter from the robot such that it faces the front of the robot. Make sure the switch on the IR transmitter is in long range mode (to the left).
4. Turn the computer on.
5. Turn the robot on by pressing the on-off button. If the robot is on, the display should not be blank.

Adjusting the microphone

The microphone should be configured before starting.

1. Choose Start -> Settings -> Control Panel ->Speech
2. Choose Configure Microphone.
3. Follow the instructions.

It is possible to make your own speech profile, although this was not done in the development of this project.
1. Choose Start -> Settings -> Control Panel -> Speech
2. Follow the instructions in Recognition Profiles.

Starting the application

1. Run scrap.exe

This window will now pop up:

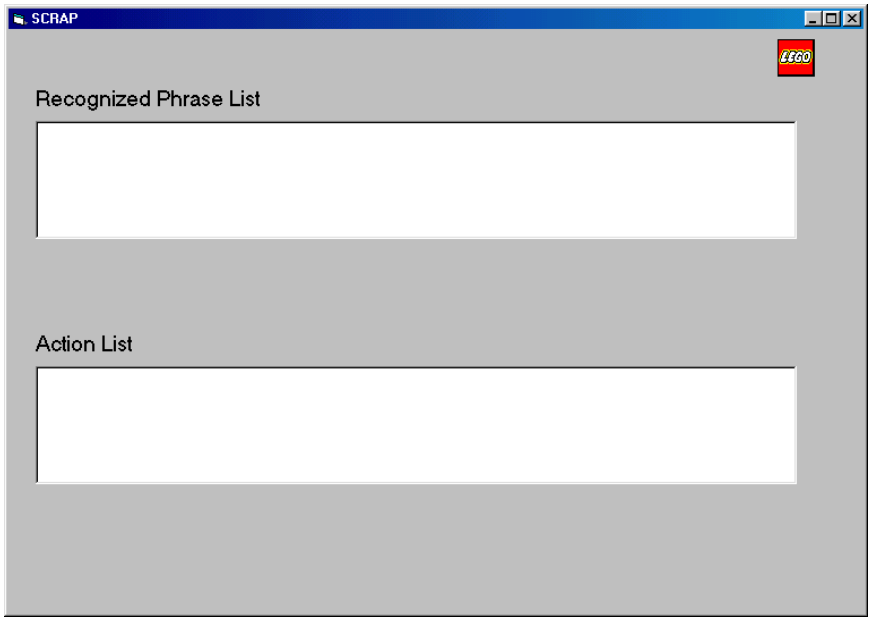

Figure A.1

All recognized phrases will be written to Recognized Phrase List. All actions by the robot will be written to Action List. If the phrase was not understood, "No recognition" will be written to both Recognized Phrase List and Action List.

You are now ready to use SCRAP.
The first thing you should do is downloading a program from the PC to the robot, see next paragraph. Some advises on speaking is given in the last paragraph.

Downloading robot program

For all functionality to work, a special program needs to be downloaded from the PC to the robot.
The first thing you should do is downloading the program:
Speak: *"my robot download program"*

If your uttered phrase was recognized correctly, the program will be downloaded to the robot. You will receive the message "Program downloaded to slot 5". On the Recognized Phrase List the phrase will be written (or more correctly, the phrase the computer *thought* it heard).
If the system didn't understand your phrase, "No recognition" will be written.

The robot computer has five program slots. If you want to download the program to a different slot, you can speak for instance: *"my robot download program to slot one"*

Moving and stopping robot

Speak: *"my robot drive forward"*
The robot should now start to drive forward.
Speak: "*my robot stop*"
The robot should now stop.
Speak: *"my robot drive back"*
The robot should now drive back.
Speak: *"my robot move forward"* or simply *"my robot forward"*
You will now see that there are alternative ways of commanding the same action.
Stop the robot.

Speak: *"my robot fifty centimeters forward"*
The robot should now move 50 cm forward and then stop.
How do you think you can make the robot move 20 cm back?

Turning robot

Make sure the robot is standing still.
Speak: *"my robot turn left"*
The robot should now turn 90° left.
Speak: *"my robot turn right"*

Make sure the robot is moving forward.
Speak: *"my robot turn left"*
The robot should now stop, turn 90° left and move forward again.
Try this for the right side.
Try this while the robot is moving back.

Speak: *"my robot turn forty five degrees left"*
The robot should now turn 45° left.
Try this for a different angle.

Power

Make sure the robot is moving.
Speak: *"my robot set power level to one"* or *"my robot power one"*
You will now see that it is possible to control the speed by adjusting the motor power.
The power level should be between 0 and 7.

Quitting

To quit the application, simply click the cross in the upper right corner of the window.

How to speak

Some advices when speaking:

- don't speak to fast but also not to slowly
- speak clearly.
- don't speak too close to the microphone, 30-50 cm is a good distance.
- try to speak with a good English accent.

B. Robot Control Test Results

In this chapter follows the results of the robot control testing described in chapter 5.2 is shown. All phrases are spoken my “My robot” + the command. A star in front of the robot shows that the command was spoken before the RCX program was downloaded to the robot.

Command	Wanted result	Result	Comment
* “download”	Program downloaded to slot 5	Program downloaded to slot 5	OK
“download to slot four”	Program downloaded to slot 4	Program downloaded to slot 4	OK
“download to slot six”	Error message	Error message	OK
* “forward”	Robot moving forward	Robot moving forward	OK
“forward”	Robot moving forward	Robot moving forward	OK
“move back”	Robot moving back	Robot moving back	OK
* “drive thirty centimeters forward”	Error message	Error message	OK
“drive thirty centimeters forward”	Robot moved 30 cm forward	Robot moved 30 cm forward	OK
“go one meter and thirty centimeters back”	Robot moved 130 cm back	Robot moved 130 cm back	OK
“one point two meter back”	Robot moved 120 cm back.	Robot moved 120 cm back.	OK
* “right”	Error message	Error message	OK
“right”	Robot turned 90 degrees right	Robot turned 90 degrees right	OK
“turn left”	Robot turned 90 degrees left	Robot turned 90 degrees left	OK
* “move fourty five degrees right”	Error message	Error message	OK
“move fourty five degrees right”	Robot turned 45 degrees right	Robot turned 45 degrees right	OK
“power level two”	Power level set to 2	Power level set to 2	OK

C. Main Application Source Code

```
VERSION 5.00
Object = "{D6CD40C0-A522-11D0-9800-D3C9B35D2C47}#1.0#0"; "SPIRIT.OCX"
Begin VB.Form Form1
   Caption         =   "SCRAP"
   ClientHeight    =   7695
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   11340
   LinkTopic       =   "Form1"
   ScaleHeight     =   7695
   ScaleWidth      =   11340
   StartUpPosition =   3  'Windows Default
   Begin VB.ListBox phraseList
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   12
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   1560
      Left            =   360
      TabIndex        =   2
      Top             =   1200
      Width           =   10095
   End
   Begin VB.ListBox actionList
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   12
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   1560
      ItemData        =   "scrap.frx":0000
      Left            =   360
      List            =   "scrap.frx":0002
      TabIndex        =   1
      Top             =   4440
      Width           =   10095
   End
   Begin SPIRITLib.Spirit PB
      Height          =   495
      Left            =   10200
      TabIndex        =   0
      Top             =   120
      Width           =   495
      _Version        =   65536
      _ExtentX        =   873
      _ExtentY        =   873
      _StockProps     =   0
   End
   Begin VB.Label Label2
      Caption         =   "Action List"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   13.5
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
```

```
      Height          =   375
      Left            =   360
      TabIndex        =   4
      Top             =   3960
      Width           =   1455
   End
   Begin VB.Label Label1
      Caption         =   "Recognized Phrase List"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   13.5
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   375
      Left            =   360
      TabIndex        =   3
      Top             =   720
      Width           =   3135
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
'Declaring recognizer object, grammar, and program slot nr of the RCX
Public WithEvents RC As SpSharedRecoContext
Attribute RC.VB_VarHelpID = -1
Public Grammar As ISpeechRecoGrammar
Public Slotnr As Integer

Private Sub Form_Load()

    'Creating recognizer object
    Set RC = New SpSharedRecoContext

    'Creating, loading and initializing grammar
    Set Grammar = RC.CreateGrammar
    Grammar.CmdLoadFromFile "C:\matthias\vb\grammar.xml", SLODynamic
    Grammar.CmdSetRuleIdState 0, SGDSActive

    'Initializing Com port, IR transceiver and sensors
    PB.InitComm
    PB.PBTxPower 1
    PB.SetSensorType 2, 4
    PB.SetSensorMode 2, 7, 0

    'Set Slot to -1 to indicate that no program has been downloaded to RCX
    Slotnr = -1

End Sub

'In case of a false recognition
Private Sub RC_FalseRecognition(ByVal StreamNumber As Long, ByVal StreamPosition As
Variant, ByVal Result As SpeechLib.ISpeechRecoResult)
    phraseList.AddItem "<No recognition>"
    phraseList.ListIndex = phraseList.ListCount - 1
    actionList.AddItem "No recognition"
    actionList.ListIndex = actionList.ListCount - 1
End Sub


'In case of a recognition
Private Sub RC_Recognition(ByVal StreamNumber As Long, ByVal StreamPosition As Variant,
ByVal RecognitionType As SpeechLib.SpeechRecognitionType, ByVal Result As
SpeechLib.ISpeechRecoResult)
```

```
    'Declaring variables
    Dim Props() As Integer
    Dim numProps As Integer
    Dim Direction As Integer
    Dim Rotation As Integer
    Dim Distance As Integer

    'Structuring phraseInfo properties in a n*2 array Props where a pair of cells
    'contains property id and property value. numProp contains number of
    'properties and thus length of array Props.
    If Not Result.PhraseInfo.Properties Is Nothing Then
        numProps = Result.PhraseInfo.Properties.Count
        ReDim Props(numProps, 2)
        For i = 1 To numProps
            Props(i, 1) = Result.PhraseInfo.Properties.Item(i - 1).Id
            Props(i, 2) = Result.PhraseInfo.Properties.Item(i - 1).Value
        Next i
    Else
        numProps = 0
    End If

    'Write recognized phrase to phrase list
    phraseList.AddItem Result.PhraseInfo.GetText
    phraseList.ListIndex = phraseList.ListCount - 1

    'Analyzing the properies. First a select-case operation to find the rule that fired
    Select Case Result.PhraseInfo.Rule.Children.Item(0).Id

        '"Download" rule
        'numProps = 0 means no slot was specified
        'Slotnr is always one less than spoken slotnr (0 to 4 instead of 1 to 5)
        'Props(1,2) is specified slot and should be between 1 and 5
        'downloadBool is a boolean that turns true when the command is valid
        Case 101
            downloadBool = 0
            If numProps = 0 Then
                Slotnr = 4
                downloadBool = 1
            Else
                If Props(1, 2) >= 1 And Props(1, 2) <= 5 Then
                    Slotnr = Props(1, 2) - 1
                    downloadBool = 1
                End If
            End If
            If downloadBool = 1 Then
                Download (Slotnr)
                actionList.AddItem "Program downloaded to slot " & Slotnr + 1
                actionList.ListIndex = actionList.ListCount - 1
            Else
                actionList.AddItem "Error: Program slot has to be between 1 and 5"
                actionList.ListIndex = actionList.ListCount - 1
            End If

        '"Power" rule
        'Props(1,2) is specified power level and should be between 0 and 7
        Case 102
            If Props(1, 2) <= 7 Then
                PB.SetPower "02", 2, Props(1, 2)
                actionList.AddItem "Power level set to " & Props(1, 2)
                actionList.ListIndex = actionList.ListCount - 1
            Else
                actionList.AddItem "Error: Power level has to be between 0 and 7"
                actionList.ListIndex = actionList.ListCount - 1
            End If

        '"Move" rule
        'Slotnr = -1 means no program is downloaded to RCX yet and the move can not be
done
        'Props(1,1) = 208 means distance was specified
        'Props(numProps,2) indicates the specified direction (0: forward, 1: back)
        'Distance is the distanc the robot should move in cm
```

```
        Case 103
            If Slotnr <> -1 Or Props(1, 1) <> 208 Then
                PB.Off "02"
                If Props(numProps, 2) = 0 Then
                    PB.SetFwd "02"
                    Direction_string = "forward"
                Else
                    PB.SetRwd "02"
                    Direction_string = "back"
                End If
                If Props(1, 1) = 208 Then
                    'If Props(1,2) = 0, then distance format is "x meters (and y
centimeters)"
                    If Props(1, 2) = 0 Then
                        Distance = Props(2, 2) * 100
                            If Props(3, 1) = 205 Then
                                Distance = Distance + FindNumber(Props, 3)
                            End If
                    'Else if Props(1,2) = 1 then distance format is "x centimeters"
                    ElseIf Props(1, 2) = 1 Then
                        Distance = FindNumber(Props, 2)
                    'Else distance format is "x point y meter"
                    Else
                        If Props(4, 2) < 10 Then
                        Distance = Props(2, 2) * 100 + Props(4, 2) * 10
                        Else
                            Distance = Props(2, 2) * 100 + Props(4, 2)
                        End If
                    End If
                    Drive Props(numProps, 2), Distance
                    actionList.AddItem "Robot moved " & Distance & " cm " &
Direction_string
                    actionList.ListIndex = actionList.ListCount - 1
                Else
                    PB.On "02"
                    actionList.AddItem "Robot moving  " & Direction_string
                    actionList.ListIndex = actionList.ListCount - 1
                End If
            Else
                actionList.AddItem "Error: Program needs to be downloaded to robot before
this operation can be executed"
                actionList.ListIndex = actionList.ListCount - 1
            End If

        '"Turn" rule
        'Slotnr = -1 means no program is downloaded to RCX yet, and the turning can not
be done
        'numProps = 1 means no angle was specified
        'Rotation is the angle should turn in degrees
        Case 104
            If Slotnr <> -1 Then
                If numProps = 1 Then                'If angle is not specified
                    Rotation = 90                   'Set Rotation to 90 degrees
                Else
                    Rotation = FindNumber(Props, 1) 'Else set Rotation to specified angle
                End If
                Direction = Props(numProps, 2)      'Set Direction to specified direction
(left or right)
                Turn Direction, Rotation            'Turn
                If Direction = 0 Then               'If direction is right...
                    Direction_string = "right"
                Else                                'If direction is left...
                    Direction_string = "left"
                End If
                actionList.AddItem "Robot turned " & Rotation & " degrees " &
Direction_string
                actionList.ListIndex = actionList.ListCount - 1
            Else                                    'If program is not downloaded
                actionList.AddItem "Error: Program needs to be downloaded to robot before
this operation can be executed"
                actionList.ListIndex = actionList.ListCount - 1
```

```
            End If

        '"Stop" rule
        Case 109
            PB.Off "02"
            actionList.AddItem "Robot stopped"
            actionList.ListIndex = actionList.ListCount - 1
        End Select

End Sub

'Sub routine for turning robot
'Parameters:
'- Direction: val 0: right, val 1: left
'- Rotation is number of degrees the robot should turn

Private Sub Turn(Direction As Integer, Rotation As Integer)

    'Storing wanted motor directions in variables in RCX
    'var 3 and var 4 is respectively left and right motor
    'value 0 and 1 is respectively forward and back
    If Direction = 0 Then
        PB.SetVar 3, 2, 0
        PB.SetVar 4, 2, 1
    Else
        PB.SetVar 3, 2, 1
        PB.SetVar 4, 2, 0
    End If
    'Rotation value stored in var 5 (number of rotation steps the sensor should measure
before turning operation is finished)
    PB.SetVar 5, 2, Rotation / 2
    'Motor values for motor 0 stored in val 0
    PB.SetVar 0, 3, 0
    PB.SelectPrgm Slotnr
    PB.StartTask 1

End Sub

'Sub routine for moving robot a specified distance
'Parameters:
'- Direction = 0 if the robot should move forward, 1 if back
'- Distance is number of cm the robot should move
Private Sub Drive(Direction As Integer, Distance As Integer)

    'Storing wanted motor directions in variables in RCX
    'var 3 and var 4 is respectively left and right motor
    'value 0 and 1 is respectively forward and back
    If Direction = 0 Then
        PB.SetVar 3, 2, 0
        PB.SetVar 4, 2, 0
    Else
        PB.SetVar 3, 2, 1
        PB.SetVar 4, 2, 1
    End If
    'Rotation value stored in var 5 (number of rotation steps the sensor should measure
before operation is finished)
    PB.SetVar 5, 2, Distance * 3
    'Setting var 0=-1 means that vehicle should stop after operation is finished
    PB.SetVar 0, 2, -1
    PB.SelectPrgm Slotnr
    PB.StartTask 1


End Sub


'Sub routine for downloading a program to a slot in the RCX
'Parameter:
'- Slot: program slot nr the program should be downloaded to
Private Sub Download(Slot As Integer)
    PB.SelectPrgm Slot                  'Select given program slot
```

```
    PB.DeleteAllTasks
    PB.BeginOfTask 1
        ' This tasks performes a complex moving operation, like driving a specified
distance or turning
        'At the execution of this task the variables should contain these values:
        'var 0: motor values for motor 0 before execution. Format: see p. 45 in the
Spirit reference.
        '       -1 in the variable means that the robot should stop after operation is
finished.
        'var 3: direction of left motor during operation (0=forward, 1=back)
        'var 4: direction of right motor during operation (0=forward, 1=back)
        'var 5: rotation value (number of rotation steps the sensor should measure before
operation is finished)

        'Set motors to specified directions
        PB.If 0, 3, 2, 2, 0             'If var 3=0
            PB.SetFwd "0"               'Set left motor forward
        PB.Else
            PB.SetRwd "0"               'Else back
        PB.EndIf
        PB.If 0, 4, 2, 2, 0             'If var 4=0
            PB.SetFwd "2"               'Set right motor forward
        PB.Else
            PB.SetRwd "2"               'Else back
        PB.EndIf
        PB.ClearSensorValue 2           'Set rotation sensor value to 0
        PB.SetVar 6, 9, 2               'Set var 6 to sensor value
        PB.AbsVar 6, 0, 6               'Set var 6 to the absolute value of itself
        PB.On "02"                      'Set motors on
        PB.While 0, 6, 1, 0, 5          'While var 6 is less then var 5 (which means that
the operation is not finished)
            PB.SetVar 6, 9, 2           'Update var 5 to abs.val. of new rotation sensor
value
            PB.AbsVar 6, 0, 6
        PB.EndWhile
        PB.If 0, 0, 2, 2, -1            'If robot should stop after operation
            PB.Off "02"                 'Stop motors
        PB.Else                         'Else restore motor settings
            PB.StartTask 2
        PB.EndIf
    PB.EndOfTask

    'Task for restoring of motor settings to what they were before the operation
    'var 0 contains the saved motor values. Format: see p. 45 in the Spirit reference
    'To understand the algorithm, see that page
    PB.BeginOfTask 2
        PB.If 0, 0, 1, 2, 128
            PB.Off "02"
        PB.Else
            PB.On "02"
        PB.EndIf
        PB.AndVar 0, 2, 15
        PB.If 0, 0, 1, 2, 8
            PB.SetRwd "02"
        PB.Else
            PB.SetFwd "02"
        PB.EndIf
    PB.EndOfTask

End Sub


'Subroutine to find a max three digit number in the "Number" format (see grammar) from
the property structure.
'Parameters:
'- Props(): n*2 array that contains the properties
'- startProp: number of the property where the number starts
'Return value: the number
'To understand the algorithm, see the "Number" format in the grammar
Private Function FindNumber(Props() As Integer, startProp)
    prop = startProp
```

```
    If Props(prop, 2) = 1 Then
        prop = prop + 1
        If Props(prop, 1) = 201 Then
            FindNumber = Props(prop, 2) * 100
            prop = prop + 1
        Else
            FindNumber = 100
        End If
    End If
    prop = prop + 1
    Do While Props(prop, 1) = 201 Or Props(prop, 1) = 206 Or Props(prop, 1) = 207
        FindNumber = FindNumber + Props(prop, 2)
        prop = prop + 1
    Loop
End Function
```

D. Grammar Code

```
<GRAMMAR LANGID="409">

      <DEFINE>
        <ID NAME="Toplevel" VAL="110"/>
        <ID NAME="Download" VAL="101"/>
        <ID NAME="DigitRule" VAL="105"/>
        <ID NAME="Power" VAL="102"/>
        <ID NAME="Number" VAL="106"/>
        <ID NAME="TeensRule" VAL="107"/>
        <ID NAME="TensRule" VAL="108"/>
        <ID NAME="Move" VAL="103"/>
        <ID NAME="Rotate" VAL="104"/>
        <ID NAME="Stop" VAL="109"/>
        <ID NAME="NumberFormat" VAL="205"/>
        <ID NAME="TeensProp" VAL="206"/>
        <ID NAME="TensProp" VAL="207"/>
        <ID NAME="DigitProp" VAL="201"/>
        <ID NAME="Direction" VAL="203"/>
        <ID NAME="DistanceFormat" VAL="208"/>
        <ID NAME="Side" VAL="204"/>
      </DEFINE>

      <RULE ID="Toplevel" TOPLEVEL="ACTIVE">
        <P>my robot</P>
        <L>
              <RULEREF REFID="Download"/>
              <RULEREF REFID="Power"/>
              <RULEREF REFID="Move"/>
              <RULEREF REFID="Rotate"/>
              <RULEREF REFID="Stop"/>
        </L>
      </RULE>

      <RULE ID="Download">
        <P>download
                    <O>program</O>
              <O>
                to slot
                <O>number</O>
                        <RULEREF REFID="DigitRule"/>
              </O>
        </P>
      </RULE>

      <RULE ID="DigitRule">
        <L PROPID="DigitProp">
             <P VAL="0">zero</P>
             <P VAL="1">one</P>
             <P VAL="2">two</P>
             <P VAL="3">three</P>
             <P VAL="4">four</P>
             <P VAL="5">five</P>
             <P VAL="6">six</P>
             <P VAL="7">seven</P>
             <P VAL="8">eight</P>
             <P VAL="9">nine</P>
        </L>
      </RULE>

      <RULE ID="Power">
        <P>
              <O>set</O>power<O>to</O><O>level</O>
              <RULEREF REFID="DigitRule"/>
        </P>
      </RULE>

      <RULE ID="Number">
        <L PROPID="NumberFormat">
```

```
          <P VAL="1">
            <O><RULEREF REFID="DigitRule"/></O>
            hundred
            <O>and<RULEREF REFID="TensRule"/></O>
          </P>
          <P VAL="0">
            <RULEREF REFID="TensRule"/>
          </P>
    </L>
</RULE>

<RULE ID="TeensRule">
    <L PROPID="TeensProp">
        <P VAL="10">ten</P>
        <P VAL="11">eleven</P>
        <P VAL="12">twelve</P>
        <P VAL="13">thirteen</P>
        <P VAL="14">fourteen</P>
        <P VAL="15">fifteen</P>
        <P VAL="16">sixteen</P>
        <P VAL="17">seventeen</P>
        <P VAL="18">eighteen</P>
        <P VAL="19">nineteen</P>
    </L>
  </RULE>

  <RULE ID="TensRule">
    <L>
          <P>
            <L PROPID="TensProp">
                <P VAL="20">twenty</P>
                <P VAL="30">thirty</P>
                <P VAL="40">fourty</P>
                <P VAL="50">fifty</P>
                <P VAL="60">sixty</P>
                <P VAL="70">seventy</P>
                <P VAL="80">eighty</P>
                <P VAL="90">ninety</P>
            </L>
            <O><RULEREF REFID="DigitRule"/></O>
          </P>
          <P><RULEREF REFID="TeensRule"/></P>
          <P><RULEREF REFID="DigitRule"/></P>
    </L>
  </RULE>

  <RULE ID="Move">
    <O>
          <L>
            <P>go</P>
            <P>drive</P>
            <P>move</P>
          </L>
    </O>
    <O>
          <L PROPID="DistanceFormat">
            <P VAL="0">
                  <RULEREF REFID="DigitRule"/>
                  <L>
                    <P>meter</P>
                    <P>meters</P>
                  </L>
                  <O>and<RULEREF REFID="Number"/>centimeters</O>
            </P>
            <P VAL="1"><RULEREF REFID="Number"/>centimeters</P>
            <P VAL="2">
                  <RULEREF REFID="DigitRule"/>point<RULEREF REFID="DigitRule"/>
                  <L>
                    <P>meter</P>
                    <P>meters</P>
                  </L>
```

```
                </P>
              </L>
        </O>
        <L PROPID="Direction">
              <P VAL="0">forward</P>
                    <P VAL="1">back</P>
        </L>
      </RULE>

      <RULE ID="Rotate">
        <O>
              <L>
                <P>go</P>
                <P>drive</P>
                <P>move</P>
                <P>turn</P>
              </L>
        </O>
        <O>
              <P>
                <RULEREF REFID="Number"/>
                degrees
              </P>
        </O>
        <L PROPID="Side">
              <P VAL="0">right</P>
              <P VAL="1">left</P>
        </L>
      </RULE>

      <RULE ID="Stop">
        <P>stop</P>
      </RULE>

</GRAMMAR>
```