M. Shahalam

Sid the Spider

Final Report
December 7, 2010

EEL 4665 - Intelligence Machine Design Laboratory

TAs : **Mike Pridgen**
Ryan Stevens
Thomas Vermeer
Tim Martin
Devin Hughes

Instructors: **Dr. A. Antonio Arroyo**
Dr. Eric M. Schwartz

Table of Contents

Abstract	3
Executive Summary	3
Introduction	3
Integrated System	4
Mobile Platform	5
Actuation	7
Sensors	8
Behaviors	8
Experimental Layout and Results	8
Conclusion	8
Documentation	8
Appendices	8

Abstract

The purpose of this project was to create an autonomous six-legged hexapod robot that would respond to voice commands. The robot was to be able to roam about avoiding obstacles and stop when voice commands were recognized. Since voice recognition circuits or modules were complicated, the voice recognition and analysis was done on a computer. The robot and computer were able to communicate with each other using the Xbee modules, where one received information (the robot received which commands to execute and proceeded to the corresponding state) and the other sent information (the computer sent commands after analyzing the microphone input from the microphone connected to it). Many problems were encountered along the process of designing, making, and tweaking this robot. There were issues with the design and some were compensated with more complex behaviors. For obstacle avoidance, the robot used one sonar sensor. Bump switches were tried, but with the given design was not practical. The sonar sensor was able to "spin" by the robot twisting, as it was not mounted on its own dedicated servo.

The process of creating this robot started with the formulation of the idea, followed by designing the model, creating the robot from the model, and fixing the actual robot to handle the problems that arose (legs would fall apart, servos wouldn't work, etc). At the end of the project there were many lessons learned on how to make a good (better) robot. Designing, programming, debugging, and tweaking were the major tasks of this project.

Executive Summary

The robot has four parts: the processing boards for control, the servos for actuation, a sonar for information from the outside world, and a Xbee module to communicate wirelessly with another Xbee module (connected to a laptop).

The basic idea of the robot is to have a proximity sensor scanning ahead of the robot for obstacles sending the information back to the microprocessor, which then determines if the robot is able (and should) walk forward. Then the microprocessor tells a servo controller to which 18 servos are connected to execute a predetermined sequence of servo movements. The robot then moves forward. Each of the legs had three servos, and thus three degrees of freedom. The gait used for walking was the tripod sequence. Each alternate leg was part of the same tripod (three legs per tripod). While one tripod lifted and moved forward, the other stayed on the ground and rotated backward, moving the robot forward. A similar tripod-like sequence was used to turn the robot left and right, in place. One tripod lifted and rotated while the other stayed on the ground and rotated the other way.

The sonar sensor scanned ahead continuously and sent the data back to the microprocessor from its analog pin. The microprocessor had a threshold for the sonar value, determined by trial and error, that was ideal for determining if there was an obstacle ahead. Upon reaching the threshold, if the robot was to keep moving forward into the obstacle, the microprocessor tells the robot to

stop, scan left, then scan right. The scanning is done by the robot twisting its body. Then, it determines which direction has the most space (largest value from sonar sensor) and turns in that direction and starts walking. If the values are the same, it turns in an arbitrary direction.

The robot has many states it can be in, for example a walking state, a standing still state, a dancing state, a turning state, obstacle avoidance state, etc. Some states lead to other states while information from the outside world can change the robot's state as well. The sonar sensor can change the state as well as the Xbee module. When the computer sends certain bytes to the robot, those bytes that correspond to a state will change the robot's state.

The computer determines if a voice command was spoken with a C# program written for this project. The program uses the Microsoft Speech Recognition Framework to determine if the input from the connected microphone is a command added to its dictionary. Then, it opens a serial port of the Xbee module connected to the computer and sends the byte associated with the robot state desired.

Introduction

I have always wanted to make a hexapod robot and IMDL was the perfect opportunity to make one. The objective of this project is to build an autonomous hexapod robot that has 3 degrees of freedom for each leg and responds to voice commands such as "dance", "roam", and "move <direction>". I also want to try some kind of mapping with the robot, although I have been told it might not be possible, so I will leave it for the end if I have time. This is my first robot.

Integrated System

The hexapod's microcontroller is an ATMEGA328 on an Arduino Duemilanove board. It was an ATMEGA 644P on a Sanguino board with the Arduino bootloader taken off the chip but I decided it would be simpler to use the Arduino interface and code in something very close to C++. It is serially connected to the SSC-32 Servo Controller from Lynxmotion, which handles the 18 servos for the 6 legs. It will also be connected to an XBee module to allow for communication with a laptop. The laptop will send commands to the hexapod based on voice commands received by a Bluetooth microphone on the robot, which will be processed on the laptop using Windows Speech SDK. (See **Figure 1**.)

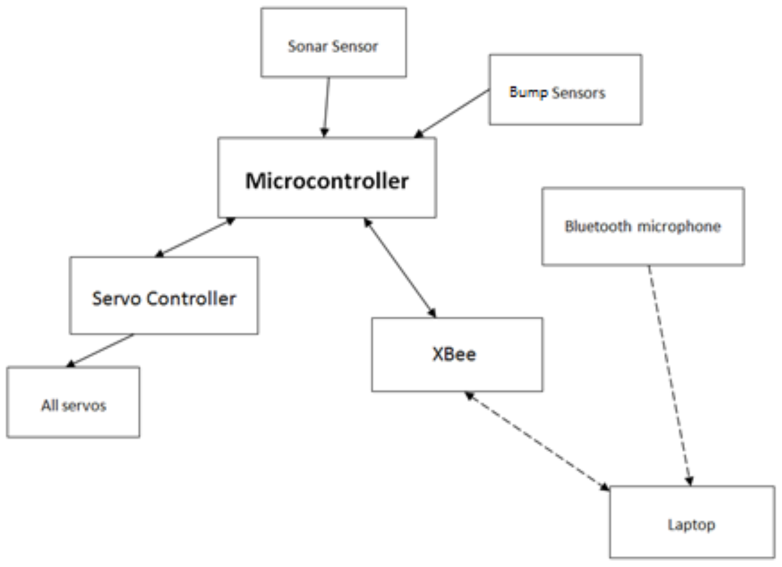

Figure 1: Flow control of the system. Arrows show direction of information flow. Dashed lines show Bluetooth communication.

Mobile Platform

The platform and the legs of the hexapod will be made from the wood provided in class. I designed the robot in SolidWorks and had the wood cut out on the T-Tech. Basically, it is an octagon shaped platform with the legs attached on the sides and the microcontroller board, servo controller, batteries, and XBee module at the center (See **Figure 2**). The legs will be 2 pieces of wood connected to 3 servos. There will be bump switches on the front, back, and possibly bottom of the robot. There is a sonar sensor in the center front. Each leg will have 1 servo attached to the platform, with another servo on top of that, with the first wood part connected to the 2nd servo. At the end of the wood, another servo would be attached to the wood, and will also connect the 2nd piece of wood. (See **Figure 3**.)

Also below is a screenshot of a SolidWorks rendering of the robot (See **Figure 4**) and a picture of the actual robot, as it is currently (See **Figure 5**).

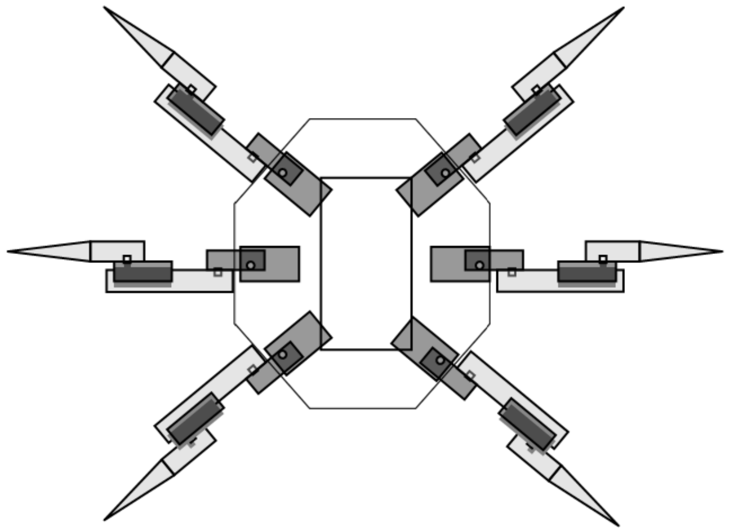

Figure 2: Preliminary design of the octagon platform with legs attached (top view). The rectangular box outlines where the boards will be, or they may go on the bottom of the platform.

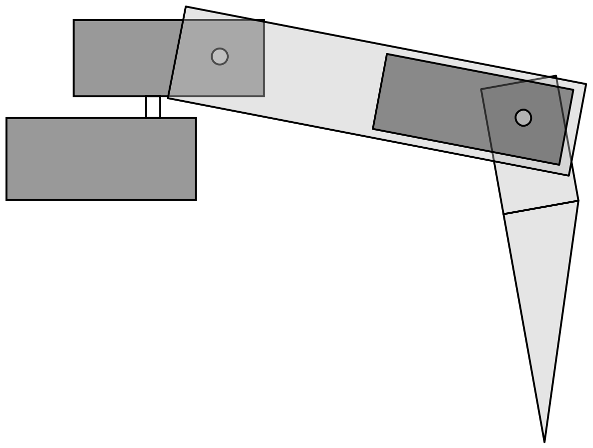

Figure 3: Preliminary design of a hexapod leg (side view), consisting of 3 servos (dark grey) and 2 pieces of wood (light grey).

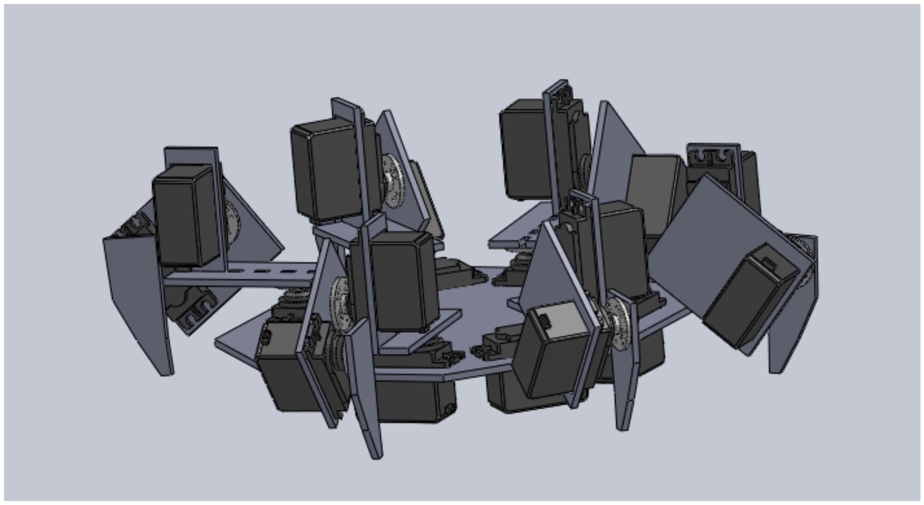
Figure 4: SolidWorks rendering of the robot, before it was T-teched.

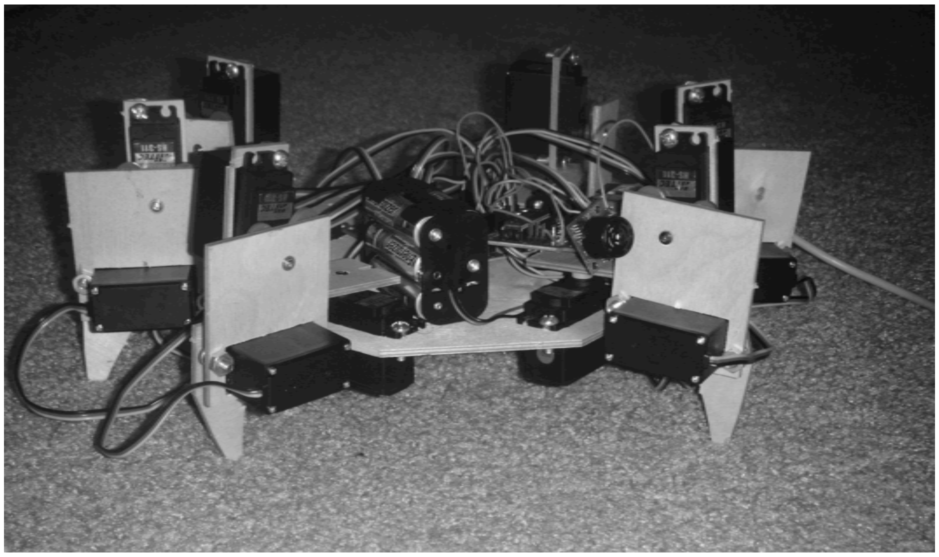
Figure 5: Actual robot.

Actuation

Actuation will be accomplished by lifting a leg, rotating it forward, and then rotating it back on the ground. After the legs have each moved forward, the servos mounted directly on the main platform will rotate, causing the robot to move forward. The legs might need some kind of rubber or other high-friction material at the bottom, so they do not slip on the floor and not move the robot.

Currently, the hexapod is using a tripod gait walking sequence to walk. It uses a similar gait to turn in place. The servo controller has the sequences necessary, and some servos are controlled individually by the microcontroller (for example, for the leg wave and twisting).

Sensors
A sonar sensor (MaxSonar-EZ2) is mounted near the center-front of the hexapod to allow the robot to avoid obstacles. Bump sensors will be mounted on the front and back, and possibly bottom as well for more input from the environment. A wireless Bluetooth microphone or headset will be used to send sounds received to a laptop, where it will be determined if someone spoke a command to the robot using the Microsoft Speech SDK, and the laptop will send via an Xbee module connected by a USB dongle to the robot's Xbee, which will tell the microcontroller what command to execute.

Behaviors
The hexapod roams around freely, avoiding obstacles, and responding to preprogrammed voice commands. The robot will walk forward from its starting position, and upon finding an obstacle, rotates in place to where there is the most space, and keep walking, repeating this algorithm. This is done by twisting the body and comparing sensor readings. The robot will listen for voice commands and upon detecting one, stop walking, and respond appropriately. Some commands currently implemented are "dance", "roam", "spin", and "move <direction>", "stop", "leg wave", and "twist".

Experimental Layout and Results

Conclusion
I have gotten the robot built working according to my own goals and specifications. It's not the prettiest robot but it does what it was supposed to do. If I had to start over, I would design this robot better, making it two-tiered and having everything done in SolidWorks in detail. I would also like to have used a PCB to fit all the boards and sensor on. However, I am very pleased with my robot and have learned much about the process of making and tweaking robots. I hope to make other robots in the future now that I have my feet wet.

Documentation

Appendices
(code)

```
#define   INIT_STATE       0
#define   START_WALKING    1
#define   KEEP_WALKING     2
#define   TURN_RIGHT       3
#define   TURN_LEFT        4
#define   DO_NOTHING       5
```

```
#define   AVOID_OBSTACLE1  6
#define   AVOID_OBSTACLE2  7
#define   WALK_BACKWARD    8
#define   TWIST            9
#define   LEG_WAVE        10
#define   DANCE           11

#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);

int state;
//int state = DO_NOTHING;

int roam = 1; // roam means no computer control

int sonarSensorPowerPin = 12;
int sonarAnalogPin = A0;

int sonarValue = 100;
int SONAR_THRESHOLD = 18;

int left = 0, mid = 0, right = 0;

void setup() {
  Serial.begin( 9600 );
   /*
  for (int i=0; i<1000; i++) {
    startingPosition();
  }
  */

  // +5V power supply for sonar sensor
  pinMode( sonarSensorPowerPin, OUTPUT );
  digitalWrite( sonarSensorPowerPin, HIGH );

  delay(1000);

  lcd.begin( 16,2 );
  //lcd.print("hello, world!");



  //startingPosition();
  //delay( 300 );

  //state = START_WALKING;
//state = DO_NOTHING;
//////////////////////////////
state = INIT_STATE;
                    //           state = TWIST;
  //initWalking();

}

void initWalking() {
  //Serial.println( "LH2000 LM1500 LL1000 RH1000 RM1500 RL2000 VS3000" );
  Serial.println( "LH2000 LM1750 LL1500 RH1000 RM1250 RL1500 VS3000" );

  //Serial.println( "LF1700 LR1300 RF1300 RR1700 HT1500" );
  Serial.println( "LF1600 LR1400 RF1400 RR1700 HT200" ); // vs = 750, ht = 750

  Serial.println( "XL100 XR100 XS100" );

}

void loop() {
  // get serial data from uP Rx
  if ( Serial.available() > 0 ) {
    // read the incoming byte
    char incomingByte = Serial.read();

    roam = 0;

    // echo
    Serial.print( " Received byte: " );
    Serial.println( incomingByte );

    switch ( incomingByte ) {
      case 'a': // walk, walk forward
        if ( state != KEEP_WALKING ) {
          state = START_WALKING;
        }
        break;
      case 'f': // stop, deactivate
        state = DO_NOTHING;
        Serial.println( "XSTOP" );
        break;

      case 'c': // turn left
        state = TURN_LEFT;
        break;

      case 'd': // turn right
      case 'e': // turn, spin, spin around
        state = TURN_RIGHT;
        break;

      case 'g': // roam
        if ( state != KEEP_WALKING ) {
          state = START_WALKING;
        }
        roam = 1;
        break;

      case 'h': // initial position
        state = INIT_STATE;
        break;

      case 'b': // walk backward
        state = WALK_BACKWARD;
        break;

      case 'i': // twist
        state = TWIST;
        break;

      case 'j': // dance
        state = DANCE;
        break;

      case 'k': // leg wave
        state = LEG_WAVE;
        break;

      default:
        break;

    }

  }

  //return;


  keepLowerLegsStill();

  sonarValue = analogRead( sonarAnalogPin );

  switch ( state ) {
    case INIT_STATE:
      lcd.clear();
      lcd.print( "initial state");
      startingPosition();
      break;

    case START_WALKING:
      lcd.clear();
      lcd.print( "starting to walk");
      initWalking();
      state = KEEP_WALKING;
      break;

    case KEEP_WALKING:
      lcd.clear();
      lcd.print( "walking");
      //lowerLegsStayStill();
      break;

    case TURN_RIGHT:
      lcd.clear();
      lcd.print( "turning right");
      turnRight();
      break;

    case TURN_LEFT:
      lcd.clear();
      lcd.print( "turning left");
      turnLeft();
      break;

    case DO_NOTHING:
      lcd.clear();
      lcd.print( "stopping");
      state = INIT_STATE;
      break;

    case AVOID_OBSTACLE1:
      lcd.clear();
      lcd.print( "avoiding obstacle");
      turnRight();
      break;

    case AVOID_OBSTACLE2:
      lcd.clear();
      lcd.print( "avoiding obstacle");
      turnLeft();
      break;

    case WALK_BACKWARD:
      lcd.clear();
      lcd.print( "walking backwards" );
      walkBackward();
      break;

    case TWIST:
      twist();
      break;

    case LEG_WAVE:
      legWave();
      break;

    case DANCE:
      dance();
      break;

    default:
      state = START_WALKING;

  }
```

```
  if ( sonarValue <= SONAR_THRESHOLD ) {
    if ( state == START_WALKING || state == KEEP_WALKING ) {
      /*if ( sonarValue % 5 ) {
        state = AVOID_OBSTACLE2;
      } else {
        state = AVOID_OBSTACLE1;
      }*/

      int dir = twistTest();
      switch ( dir ) {
        case 0: state = AVOID_OBSTACLE2;
                break;
        case 1: state = START_WALKING;
                return;
                break;
        case 2: state = AVOID_OBSTACLE1;
                break;
        default: break;
      }

    }
  }
  else {
    if ( state == AVOID_OBSTACLE1 || state == AVOID_OBSTACLE2 ) {
      // stop turning right, time to walk straight again
      Serial.println("XL100 XR100 XS100");
      state = KEEP_WALKING;
    }
  }

  //startingPosition();

 //Serial.println("         SONAR VALUE: " );
 //Serial.println( sonarValue );

}

void twist() {
  startingPosition();
  delay( 100 );

  Serial.println( "#1 P1600 #3 P1600 #5 P1600 #17 P1600 #19 P1600 #21 P1600" );
  delay( 100 );

  Serial.println( "#1 P1400 #3 P1400 #5 P1400 #17 P1400 #19 P1400 #21 P1400" );
  delay( 100 );

  state = INIT_STATE;

}

int twistTest() {


  startingPosition();
  delay( 100 );

  lcd.clear();
  mid = analogRead( sonarAnalogPin );
  lcd.print( sonarValue );


  Serial.println( "#1 P1600 #3 P1600 #5 P1600 #17 P1600 #19 P1600 #21 P1600" );
  delay( 100 );

  lcd.clear();
  left = analogRead( sonarAnalogPin );
  lcd.print( sonarValue );

  Serial.println( "#1 P1400 #3 P1400 #5 P1400 #17 P1400 #19 P1400 #21 P1400" );
  delay( 100 );
  lcd.clear();
  right = analogRead( sonarAnalogPin );
  lcd.print( sonarValue );

  if ( mid >= left && mid >= right ) return 1;
  else if ( left >= mid && left >= right ) return 0;
  else return 2;

}

void legWave() {
  Serial.println( " XSTOP " );

  startingPosition();

  // raise each leg, lower each leg.

  //Serial.println( "#0 P1500 #2 P1500 #4 P1500 #16 P1500 #18 P1500 #20 P1500" );
  Serial.println( "#0 P1000" );
  delay( 50 );
  Serial.println( "#0 P1500" );

  Serial.println( "#2 P1000" );
  delay( 50 );
  Serial.println( "#2 P1500" );

  Serial.println( "#4 P1000" );
  delay( 50 );
  Serial.println( "#4 P1500" );

  Serial.println( "#16 P2000" );
  delay( 50 );
  Serial.println( "#16 P1500" );

  Serial.println( "#18 P2000" );
  delay( 50 );
  Serial.println( "#18 P1500" );

  Serial.println( "#20 P2000" );
  delay( 50 );
  Serial.println( "#20 P1500" );

  state = INIT_STATE;
}

void dance() {
  twist();
  twist();
  legWave();
  twist();
  legWave();
  twist();

  state = INIT_STATE;

}

void turnRight() {
  initWalking();
  Serial.println( "XL-100 XR100 XS100" );
}

void turnLeft() {
  initWalking();
  Serial.println( "XL100 XR-100 XS100" );
}

void walkBackward() {
  initWalking();
  Serial.println( "XL-100 XR-100 XS100" );
```

```
}

void startingPosition() {
  Serial.println( "XSTOP" );

  // horizontal legs all starting position
  Serial.println( "#1 P1500 #3 P1500 #5 P1500 #17 P1500 #19 P1500 #21 P1500" );

  // verticle legs all starting position
  Serial.println( "#0 P1500 #2 P1500 #4 P1500 #16 P1500 #18 P1500 #20 P1500" );

  // lower legs all starting position
  Serial.println( "#9 P1500 #10 P1500 #11 P1500 #25 P1500 #26 P1500 #27 P1500" );

}

void keepLowerLegsStill() {
  // lower legs all starting position
  Serial.println( "#9 P1500 #10 P1500 #11 P1500 #25 P1500 #26 P1500 #27 P1500" );

}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Speech.Recognition;
using System.Speech.Synthesis;
using System.IO.Ports;

namespace SpeechRecognition
{

    public partial class Form1 : Form
    {
        SpeechRecognizer rec = new SpeechRecognizer();
        SpeechSynthesizer speak = new SpeechSynthesizer();

        SerialPort serialPort; // = new SerialPort();

        public Form1()
        {
            InitializeComponent();
            rec.SpeechRecognized += rec_SpeechRecognized;
            serialPort = new SerialPort("COM8", 9600);
            serialPort.Open();

            if (!serialPort.IsOpen) MessageBox.Show("Error: Serial Port was not opened");
        }

        void rec_SpeechRecognized(object sender, SpeechRecognizedEventArgs e)
        {
            //MessageBox.Show("Speech recognized! " + e.Result.Text);
            lblLetter.Text = e.Result.Text;

            executeCommand(e.Result.Text);
        }

        private void executeCommand(string command)
        {
            speak.Speak(command);

            switch (command)
            {
                case "walk":
```

```
        case "walk forward":
        case "activate":
        case "forward":
            send('a');
            break;
        case "walk backward":
        case "backward":
            send('b');
            break;
        case "turn left":
        case "left":
            send('c');
            break;
        case "turn right":
        case "right":
            send('d');
            break;
        case "turn":
        case "spin":
        case "spin around":
            send('e');
            break;
        case "stop":
        case "deactivate":
            send('f');
            break;
        case "roam":
        case "roam free":
            send('g');
            break;
        case "initial position":
            send('h');
            break;
        case "twist":
            send('i');
            break;
        case "dance":
            send('j');
            break;
        case "leg wave":
            send('k');
            break;

        default:    // part of dictionary, but not implemented yet
            send('x');
            break;
    }
}


private void send(char message)
{
    if (serialPort.IsOpen)
    {
        serialPort.Write(message.ToString());
    }
}

void Form1_Load(object sender, EventArgs e)
{
    var c = new Choices();
    c.Add("spin");
    c.Add("walk");
    c.Add("walk forward");
    c.Add("walk backward");
    c.Add("turn around");
    c.Add("spin around");
    c.Add("turn");
    c.Add("turn left");
```

```
            c.Add("turn right");
            c.Add("stop");
            //c.Add("start");
            c.Add("activate");
            c.Add("deactivate");
            c.Add("help");
            c.Add("dance");
            c.Add("roam");
            c.Add("roam free");
            c.Add("initial position");

            c.Add("left");
            c.Add("right");
            c.Add("forward");
            c.Add("backward");

            c.Add("twist");
            c.Add("dance");
            c.Add("leg wave");

            var gb = new GrammarBuilder(c);
            var g = new Grammar(gb);
            rec.LoadGrammar(g);
            rec.Enabled = true;
        }

        private void lblLetter_Click(object sender, EventArgs e)
        {

        }

        private void button6_Click(object sender, EventArgs e)
        {
            executeCommand("initial position");
        }

        private void button1_Click(object sender, EventArgs e)
        {
            executeCommand("walk forward");
        }

        private void button2_Click(object sender, EventArgs e)
        {
            executeCommand("walk backward");
        }

        private void button5_Click(object sender, EventArgs e)
        {
            executeCommand("stop");
        }

        private void button3_Click(object sender, EventArgs e)
        {
            executeCommand("turn left");
        }

        private void button4_Click(object sender, EventArgs e)
        {
            executeCommand("turn right");
        }

        private void button7_Click(object sender, EventArgs e)
        {
            executeCommand("twist");
        }

        private void button8_Click(object sender, EventArgs e)
        {
            executeCommand("dance");
        }

        private void button9_Click(object sender, EventArgs e)
        {
            executeCommand("leg wave");
        }
    }
}
```