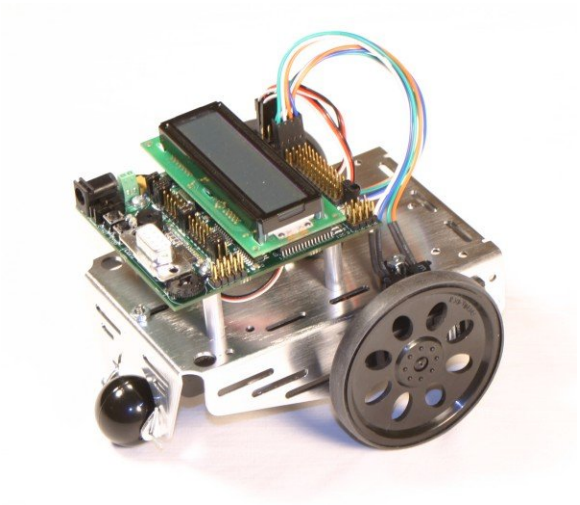
Neven Skoro and Jonathan Eren

A Study on Robotics

COP-4908

Instructor: Janusz Zalewski

April 5, 2006



IntelliBrain Robot

UMI RTX Robot

1. Introduction and Overview

The robot, a programmable machine that is designed to move and act based on what the programmer predefines. The purpose of a robot can vary from doing a repetitive task to exploring new areas in space. The most widely used robot is an industry robot; they are mostly made to do the repetitive tasks, such as welding or cutting. It is more beneficial for a company to use an industry robot seeing as they can work the same jobs as humans but with more precision and speed. A robot can also reach places that are difficult for a human to go to or is too dangerous for a human such as going into a volcano or disposing of a bomb. Robots are also developed for their ability to do services to humans such as handicapped assistants. As the technologies evolve newer robots are being developed that have abilities to see, hear, and even make decisions. The RTX robot has 6 degrees of freedom and a gripper. This robot was originally developed as an educational tool but from the RTX another robot that was used in the industry was created from it.

The first degree of freedom is the movement up and down of the arm, thus it is linear. The second (shoulder) and third (elbow) degrees focus on the rotation of the arm. Continuing down the arm of the robot is the wrist which contains 3 degrees of freedom which control the yaw, pitch, and roll of the wrist. The RTX is designed to work in a cylindrical workspace where vertical movements and rotation around the base can be easily preformed. This robot was initially designed to be programmed in Turbo Pascal and had libraries to support it. Using a basic serial link from the computers RS-232 socket a connection with the robot was made where binary commands can be sent

directly to control the arm. To improve upon the RTX a program to control it in Java was developed and will be explained in this report.

Most of the robots built today are two-wheel differential meaning that all of the robots movement is accomplished with just one set of wheels. The IntelliBrain bot is composed of the IntelliBrain main board that takes the energy from 4 AA batteries to power the two servo motor ports as well as other sensor ports. This robot has the following components attached to it's main board: an LCD display a couple of push buttons, a beeper, an SRF04 sonar range sensor; two wheels that are attached to the servo motors, a set of GP2D12 range sensors and two QRB1134 infrared photo reflector sensors used for wheel encoding as well as line sensing. All of this hardware sits on a well-built Boe-Bot chassis developed by Parallax.

The robot runs on java virtual machine that supports multi threading which is essential for this project because we need to have multiple objects running at the same time. This IntelliBrain bot uses RoboJDE Java development environment software that is used to design, build and upload the program to the main board of the robot. This software comes with the API specification of classes and methods that have already been developed for public use. The program is uploaded to the robot via a serial cable.

The robot's main board consists of 10 Digital inputs and outputs, an infrared receiver, 3 IC ports, a buzzer, a 16x2 LCD display, 2 servo ports, 7 analog/digital inputs, a power switch, 2 push buttons, a thumbwheel, a host Interface (RS-232, 115.2K baud) and a wall brick power connector. The robot runs on an Atmel Atmega128 Micro controller that has 14.7 MHz, 128 flash program memory, 132K RAM and 4K bytes of EEPROM. The API specification notes that the flash memory has a limited life of 10,000

writes and therefore should be used sparingly, while the 4K of EEPROM is estimated to have a lifetime of 100,000 writes. The Java Virtual Machine and the bootstrap loader use 4K bytes of Random Access Memory (RAM). The infrared receiver works at 38 kHz and is compatible with most television remotes; this receiver could also be used for communication between two robots. The bootstrap leader is software on the robot responsible for downloading the virtual machine to the robot and changing the host port baud rate.

2. Problem Formulation

2.1 - RTX

2.1.1 RTX Communication Overview

The communication with the robot and the PC is accomplished via a serial cable from the RS232 socket. The robot's motors operate at 62.5 Hz.

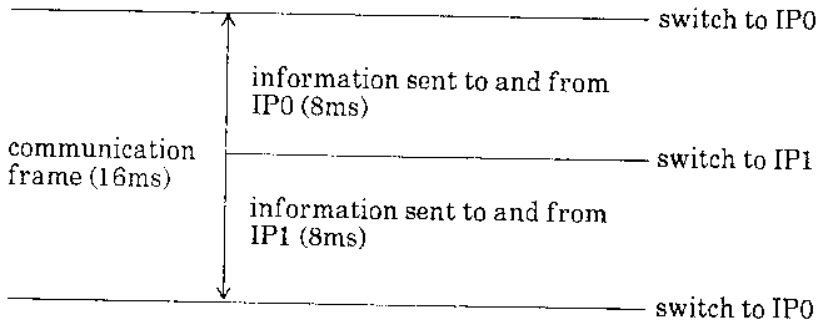


Figure 2.1 – communication frame

The communication frame between the computer and the robot is 16 milliseconds as seen from the figure. This means that there are 8ms to send information to the robot and 8ms to receive the information from the robot. This RTX robot has two intelligent peripherals (IP), IP0 and IP1 that control certain motors in the robot. The problem is sending the

write combination of bytes to the corresponding IP in order to move a certain motor. Furthermore, the program should be able to determine the home position of each motor and the assigned location.

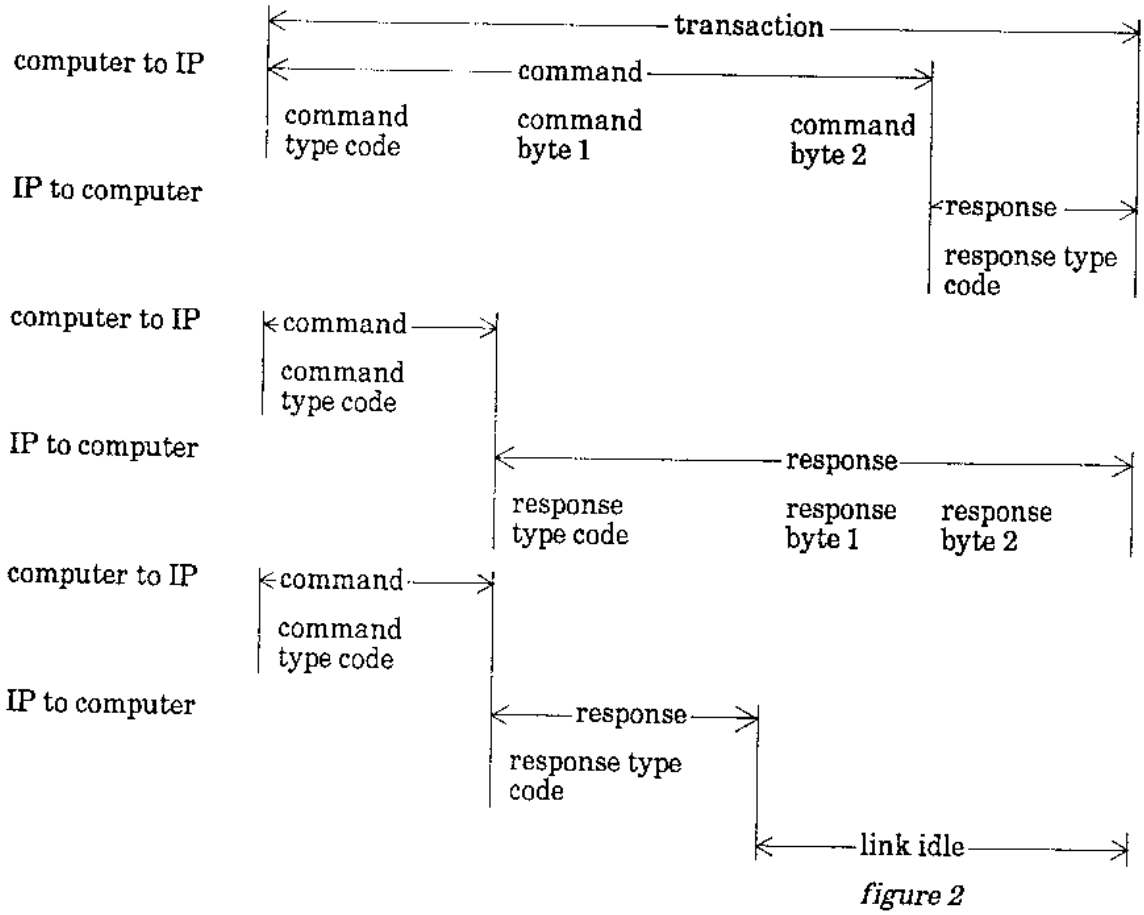


Figure 2.2 – three possible transactions

From the figure 2.2 we can see that each transaction consists of four bytes therefore there are three different ways we can communicate with the robot. First option is to send three commands the first being the type command code and the other two being byte 1 and byte 2 and receive one response. Second option is to send one command and receive three responses and the third option is to send a command type code and receive a response type code. The problem is figuring out what set of bytes the robot needs to receive in

order to move the specified motor. Furthermore, it needs to be assured that the user does not send too many commands and clog the robot buffer. And finally, the response which the robot gives needs to be interpreted and addressed accordingly.

2.1.2 Requirements Overview

The robot GUI needs to be created in order to operate the robot arm. This GUI needs to be able to accept different inputs for different motors. The biggest issue is dealing with how the robot understands negative numbers. These numbers have to be converted into the corresponding bytes that the robot can understand.

2.1.2.1 Specific Software Requirements

The communication between the RTX robot and the PC cannot exceed 16 milliseconds.

2.1.2.3 Input Data Requirements

a) A five bit supervisory code is received from the robot for every command involving motor modes, status requests and emergency stops

b) A five bit parameter code is received from the robot for every parameter command issued to the robot to ensure the correctness of the command sent.

2.1.2.3 Output Data Requirements

a) A three bit code has to be sent to the robot to specify which IP should be used.

b) A five bit code is sent for referring to the control parameters.

2.2 IntelliBrainBot

2.2.1 Requirements Overview

There is a need for a robot explorer such as the moon or mars explorer. This robot should be independent and self aware of the environment around it. Design a program that will run the robot from location A to location B while memorizing its path and avoiding obstacles. The primary objective is design an algorithm that will provide the robot with enough information about the environment that it can travel on its own without bumping into things. While running multiple classes this algorithm has to give certain threads the maximal priority or certain sensors will miss their deadline and the robot could hit an object or fall down a whole or a canyon.

This is a real time system and it has bound response time, it is crucial that there are no defects in the algorithm and the code itself. Sensors will only be used when needed in order to save energy.

2.2.1.1 General Functionality and Load Requirements

a) Using sonar sensors the robot shall know if it is close to an object

b) The robot shall display various warning signs on the LCD screen

c) The robot shall display the distance to the obstacle if it encounters one using range sensors

d) The robot shall produce a beeping noise if it encounters an obstacle such as a wall.

e) It shall move away from an obstacle if it is close to it as 5 cm

f) It shall be able to go around an obstacle

g) The robot shall memorize its path and navigate using dead reckoning

h) Using different power on separate motors the robot shall move in more sophisticated movements such as arcing.

i) The robot shall detect holes in the path and avoid them

j) The robot shall accept commands from a Universal remote controller via infrared signal

2.2.1.2 Input Data Requirements

- It shall allow the use of two buttons for start and stop of the program
- Using the range sensors it shall input the distance data and convert it to centimeters
- Shaft encoders shall register high or low voltage when sending signals to the wheels
- It shall input the data from shaft encoders of the number of wheel rotations
- The robot shall register user inputs from a universal remote controller

2.2.1.3 Output Data Requirements

- The controller shall display various data and warnings on the LCD screen
- Range sensors shall output the infrared signal to the environment
- Shaft encoders shall send out infrared beams to the wheels
- The controller shall send the signal to the servomotors to calibrate the power

2.2.1.4 Joint Input/Output Data Requirements

- The controller shall send and receive the short burst of sound from the sonar sensors

2.2.1.5 Detailed Functional Requirements

a) Sonar sensor shall send a signal and receive an echo from an obstacle, it shall then convert the analog sound to digital, analyze the data and call the methods getdistance() to calculate how far the obstacle is.

b) Different sensors shall send data to the LCD screen in a format of two lines, and the controller shall have a class that contains several screen objects, which are called via the update method. Java's multi threading capabilities shall come in handy here because the robot shall be updating the screen while executing other operations.

c) Display different information on the LCD screen shall get the minimum priority for they are not as crucial to be executed on time

d) Range sensors use infrared beams to determine the distance of an object under the exception of being exposed to direct sunlight or when facing reflective glittering surfaces where the sensors shall show inaccurate data.

e) By controlling servos control circuitry we can adjust different power levels to each of the servos thus enabling the robot to curve and not just move straight.

f) Calculations are used to determine if the robot should reverse and turn in order to go around an obstacle, it shall try to avoid an obstacle by moving 90 degrees to the right for 3 seconds and then repeating the process

g) When using dead reckoning the robot shall be able to keep track of Y, X and Theta variables on a graph. When the wheels rotate forward a counter shall add up and when the wheels rotate backwards the counter shall reduce itself. Calculation of the counts per rotation on the other had shall be calculated by dividing the wheel diameter by the track width and multiplying it with the counts per revolution.

3. System Design

3.1 Context Diagram for RTX

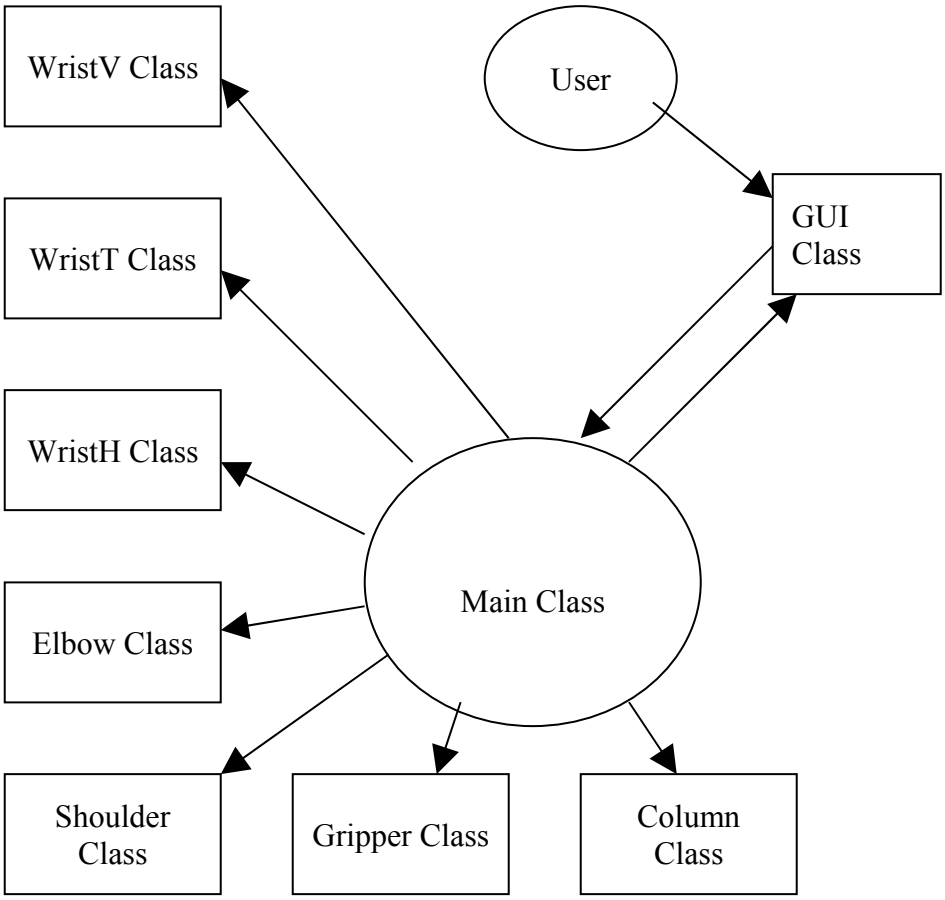


Figure 3.1 – RTX context diagram

The RTX program is controlled by the main class which initializes all the motors and communicates with the user via the GUI class. Each motor is controlled by a separate class. When the user enters the information to the GUI, the GUI calls the methods in the main class which operates on separate motor objects created.

3.2 Context Diagram for IntelliBrain Bot

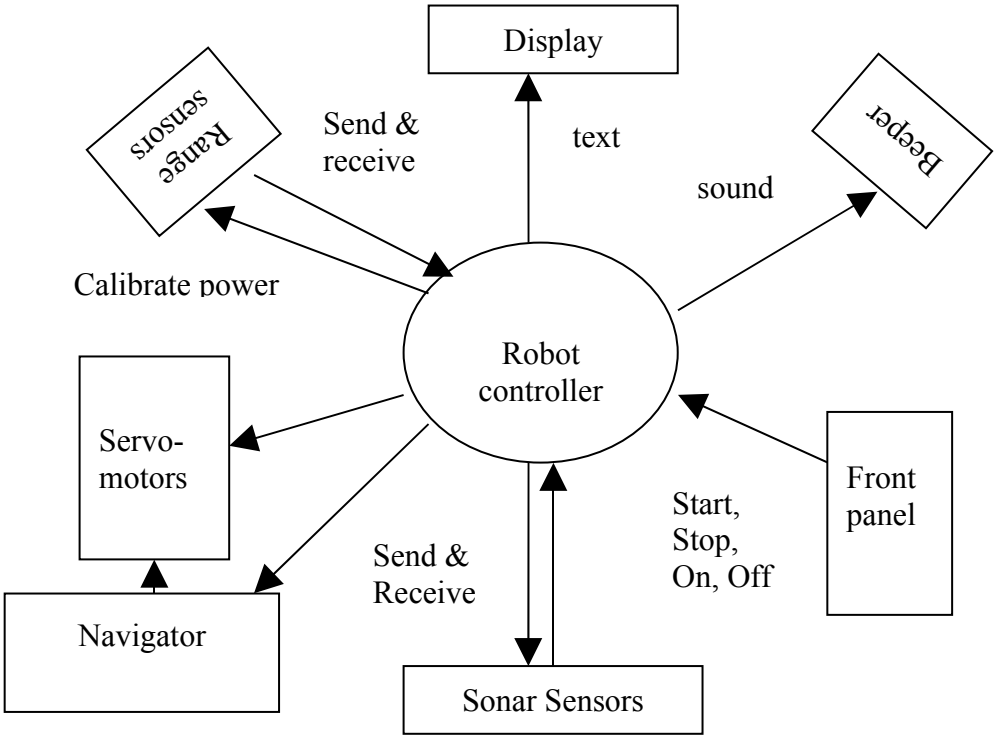


Figure 3.2 – IntelliBrain context diagram

The robot controller shall communicate with multiple classes shown below. It shall send warnings and data to the Display screen; it shall receive inputs from the front panel for start and stop or to scroll through different functions. It shall communicate with the 2 range sensors and the sonar sensor back and forth. The range sensors and the sonar sensor shall communicate with the robot controller and report any data that indicate there are obstacles in the way. The navigator class shall send information to the servos of how much power to use in order to turn in the right direction and thus navigate towards the destination.

4. Implementation

4.1 Implementation for RTX robot

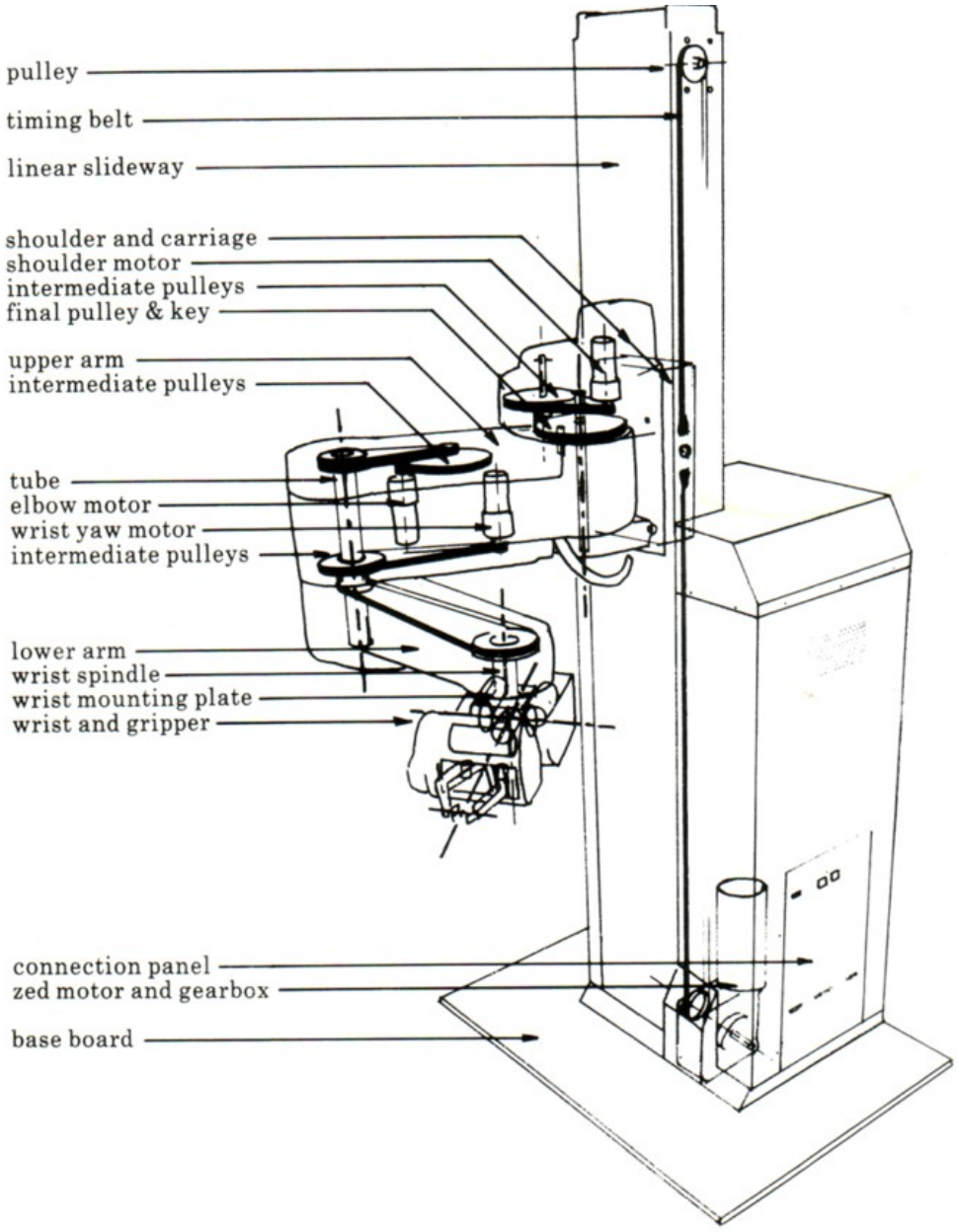


Figure 4.1 – RTX parts layout

Both IPs are Intel 8031 micro controllers that have a proportional, integral and differential control algorithm and velocity profiling. Intel 8156 256-byte RAM, 27128 EPROM containing each IP's 16-Kbyte firmware and an 8243 input / output expander are located on each IP. All of the motors are driven by the Sygnetics L293 chips giving out about one third of an amp to each motor.

There are exactly seven 24V DC motors in the RTX robot. The vertical motion is controlled via a 20W output power motor. The other six motors are of 6W output power and they control the other axes. Similarly to the IntelliBrain bot, the RTX robot's motors have two encoders each to control the distance moved of each motor. These encoders are two phase optical incremental encoders that have maximum and minimum values specified differently for each motor. The RTX firmware counts all of the state changes for all the encoders in each motor. Both the direction and the position of the motor can be determined by using the two encoders.

When the robot is initialized in its home position all of the motor encoders are set to zero. All of the motors have 12 encoder counts per motor revolution except for zed which has 24. The separation of the two gripper halves in millimeters is measured by the following equation:

S = (0.0584*c) + (10.7*10-6*c2)

(*c is the number of encoder counts registered by the gripper encoder.)

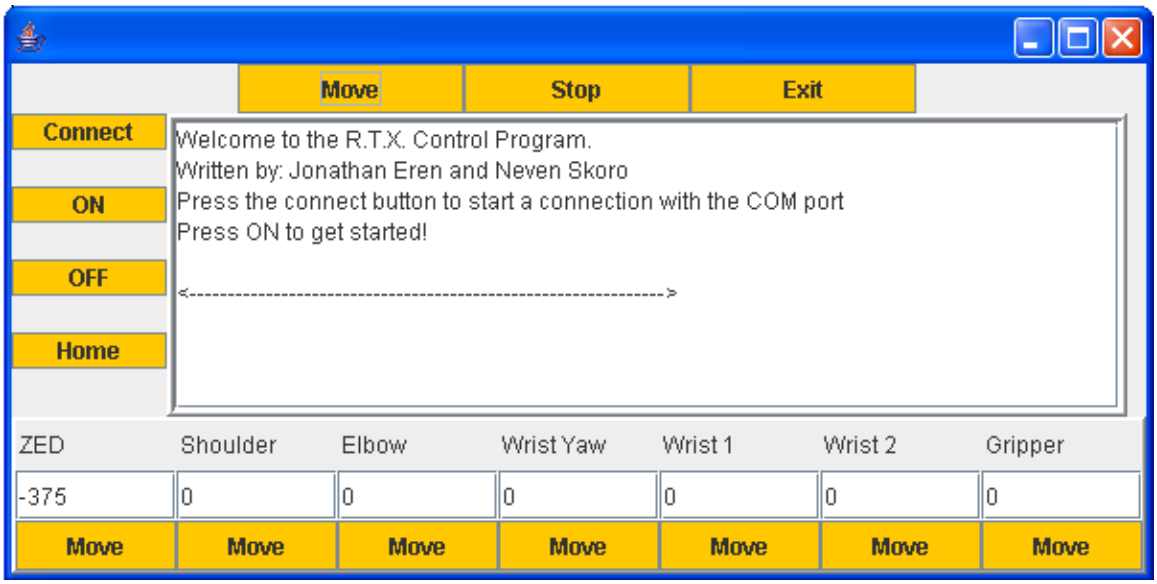


Figure 4.2 – GUI layout

A GUI written in Java was developed using the applet ability (Fig 4.2). The GUI is developed with the 6 degrees off freedom in mind each column is for one of the motors. The text fields are for which way the user wants to move the motor. Each motor is controlled by an encoder count with a maximum and a minimum which that motor can move. Each column has its own text field that can move that specific motor or a general move button which takes all the different encoder counts and moves the ones that are different. The connect button uses the javax.comm.*; api to find an active communications port on the computer that uses a serial link on the RS-232 socket. By creating that connection the commands can be sent to the robot. The robot needs to be initialized when it turns on, thus the ON button initializes the robot once the connection has been made. Once the connection to the robot is no longer needed the OFF button cancels the link to the robot by closing the COM port on the computer. A default home position for the robot has been predefined with specific encoder counts; the Home button

automatically takes the robotic arm to that default position and changes the values in the text fields to the appropriate encoder counts. The robotic arm has 3 different kinds of stops, dead stop, which stops the motor instantly and then locks in place to prevent it be moved by hand, ramp stop which gradually stops the motors but still locks them in place as well, and finally free stop which stops the motors but allows manual movement of the arm. For simplicity of the program no stop ability has been placed in this program. The visual layout of the GUI was made by a border layout for the frame and then grid layout panels were added to the frame.

The encoder counts that are sent to the robot have a predefined maximum and minimum to prevent the user from moving the arm beyond its capabilities. The maximum ranges of the motors are as follows:

- Column: 0 to -3550
- Shoulder: 2633 to -2630
- Elbow: 2278 to -2645
- Wrist Yaw: 846 to -846
- Wrist 1: -1326 to 0
- Wrist 2: 1195 to -1195
- Gripper: 1200 to 0

The arm responds to these encoder counts and shall move the robot accordingly the GUI shall help in preventing the user from making the motor move beyond its limits. This was written to be a replacement for the Turbo Pascal libraries that the robot was initially programmed in.

Sample code:

this method sends bytes to the robot IPs

```
public static void pushIt(byte b[]) {

      try {
         outputStream.write(b);
         outputStream.flush();
      } catch (IOException ex) {
         System.out.println(ex.toString());
      }

// this method converts integers to bytes

public static byte[] intToBytes(int i) {

      b = new byte[2];
      int i1, i2;

      if (i == 0) {
         b[0] = 0;
         b[1] = 0;
      }
      if (0 < i & i <= 255) {
         b[0] = (byte) i;
         b[1] = 0;
      }
      if (i > 255) {
         b[0] = (byte) (i % 256);
         b[1] = (byte) (i / 256);
      }
      if ( -255 <= i & i < 0) {
         b[0] = (byte) (255 + i);
         b[1] = (byte) 255;
      }
      if (i < -255) {
         b[0] = (byte) (255 + (i % 256));
         b[1] = (byte) (255 + (i / 256));
      }
      return b;
   }
```

4.2 Implementation for IntelliBrain robot

The program consists of the main class called MyBot, which creates objects of all other classes. Multiple threads shall be running at the same time with priority given to sensor threads and navigation. It is important that the robot samples data from the 3 distance sensors frequently enough so it does not bump into objects or fall down a hole, stairs, canyons, etc. These sensors and the navigation classes shall communicate with the MoveRob class that is responsible for simple movements such as forward, backward, left and right assigned by the user via a remote controller. If the sensors “see” an obstacle in the way they shall shut down the motors and let the user know that there is an object in the way by displaying it on the LCD screen.

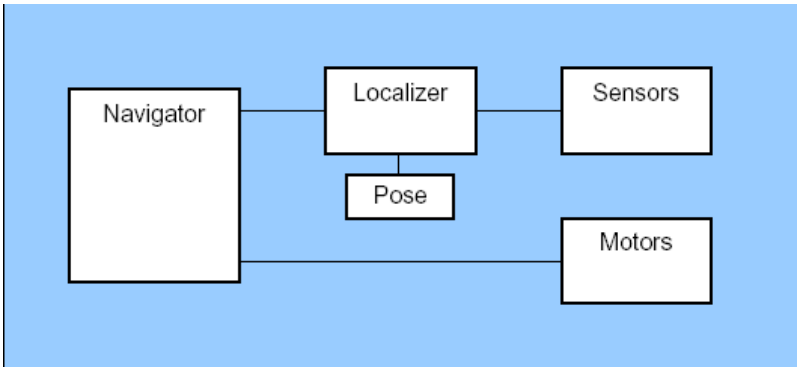


Figure 4.3 – Software design diagram

There shall be two objects created of the AnalogShaftEncoder class each for the left and right encoder. This class shall monitor and sample the data that it receives from the shaft encoders and it will be given the highest priority because if it misses the edges of the holes in the wheel it will show inaccurate data for the navigation classes so it is vital that it samples the voltages frequently.

The Localizer class will keep track of the current position and it will monitor the wheel sensors to determine the current Pose and it will let the other classes access the data through a "getter" method. This data will be available as an instance of the class Pose. The navigation class will get position data from the localizer class and it will decide how to power each motor to get to the destination. The Motors will tell the Localizer class which way the wheels are turning so the Localizer knows whether to increase or decrease the counter.

The OdometricLocalizer class will calculate the distance per count by the equation: *distancePerCount = Pi * diameterWheel / countsPerRevolution* and delta distance by: *deltaDistance = (leftCounts + rightCounts) /2.0 distancePerCount*. This class will also calculate the counts per rotation by the equation: *countsPerRotation = (trackWidth / wheelDiameter) * countsPerRevolution* as well as radians per count and

delta heading respectively *radiansPerCount = Pi * (wheelDiameter/trackWidth) / countsPerRevolution,*

*deltaHeading = (rightCounts leftCounts) * radiansPerCount.*

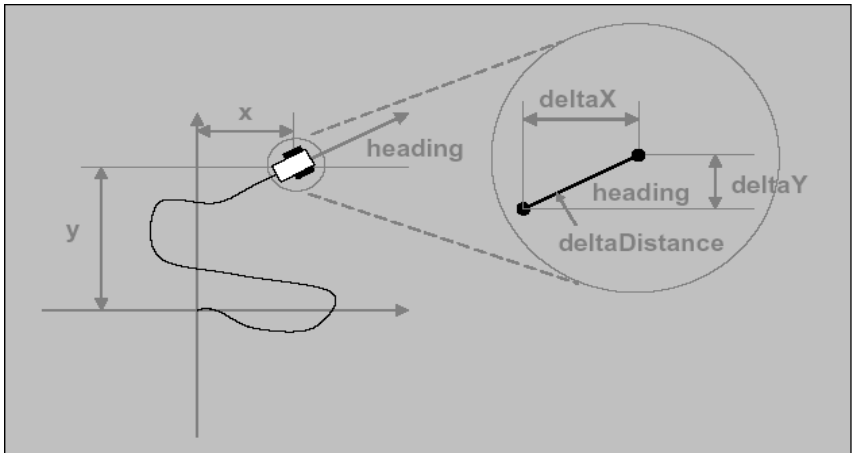


Figure 4.4 – navigation theory

Furthermore, this class will calculate the delta X and delta y by the equations: *deltaX = deltaDistance * cos(heading), deltaY = deltaDistance * sin(heading)*. The dead reckoning operations will be samples frequently enough so the robot can keep track of its position and where it's going. These dead reckonings operations can simply be done with the following code:

```
int deltaLeft = leftCounts - mPreviousLeftCounts;
int deltaRight = rightCounts - mPreviousRightCounts;
float deltaDistance = 0.5f * (float)(deltaLeft + deltaRight)* mDistancePerCount;
float deltaX = deltaDistance * (float)Math.cos(mHeading);
float deltaY = deltaDistance * (float)Math.sin(mHeading);
float deltaHeading = (float)(deltaRight - deltaLeft)* mRadiansPerCount;
```

The SRF04 sensor will frequently sample the data and stop the motors if it encounters an object that is closer than 3 inches. The 2 GPD12 sensors will sample the data and monitor big changes in distance; these sensors will be positioned at an angle of 45 degrees so they can sense holes in the path. By monitoring for huge changes in distance they will alert the main class and stop the motors from running towards the hole and then back up.

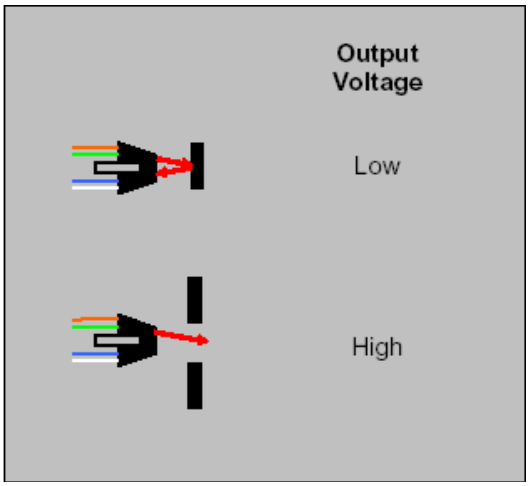


Figure 4.5 – wheel encoder voltage output

The GP2D12 will actually show that the object is really far away when it is closer than 10 cm. Furthermore, it has a maximum distance of 80 cm which is much less than the SRF04. GP2D12 sensors are analog and always turned on. The much better SRF04 sonar sensor uses 5 volts and a current of 30mA. It sends a ping at a rate of 40 KHz with a minimum range of 3 cm and a maximum of 3 meters. It is able to detect a 3cm diameter stick at a distance greater than 2 meters. Its ping shatters in different directions making it useful in detecting objects that are not just in front of it.

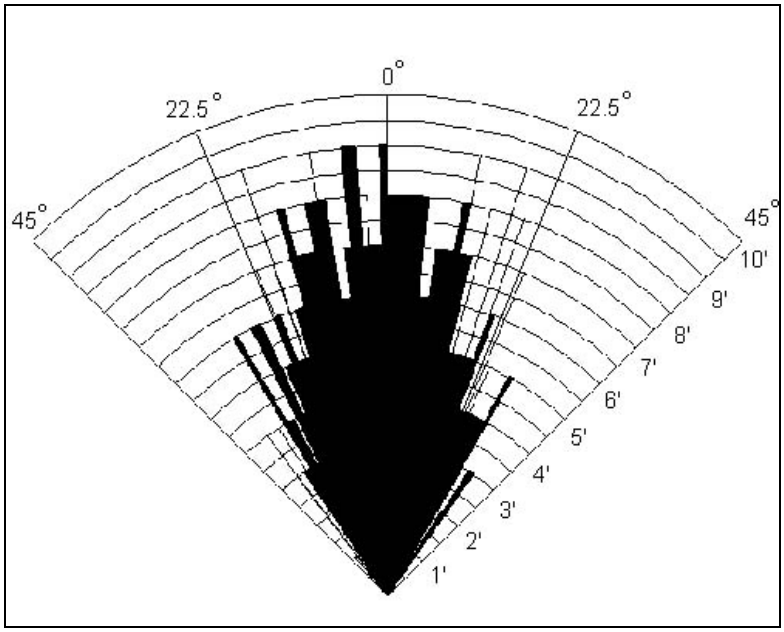


Figure 4.6 – Sensor Beam

We can see from the figure 1.4 that the beam spreads up to 22.5 degrees in both directions thus making it better at detecting objects. A lot of problems occurred when using this sensor such as Null Pointer Exceptions because of bad wiring. During the testing we have encountered numerous problems with the GP2D12 sensors because they are very sensitive of light. If there is direct light hitting the sensors they will not work properly as well as if there is not enough light. This is an issue because these 2 sensors detect holes in the path and therefore could let the robot fall down in case they are not functioning properly. Figure 1.5 shows the different maximal and minimal distances from similar Sharp range sensors.

Ranger	Output	Minimum Distance	Maximum Distance	On Current	Off Current
GP2D02	Serial	10cm	80cm	~22mA	~3mA
GP2D05	Digital	-	fixed at 24cm	~10mA	~3mA
GP2D12	Analog	10cm	80cm	~33mA	always on
GP2D15	Digital	-	fixed at 24cm±3cm	~33mA	always on
GP2D120	Analog	4cm	30cm	~33mA	always on
GP2Y0A02YK	Analog	20cm	150cm	~33mA	always on
GP2Y0D02YK	Digital	-	fixed at 80cm	~33mA	always on

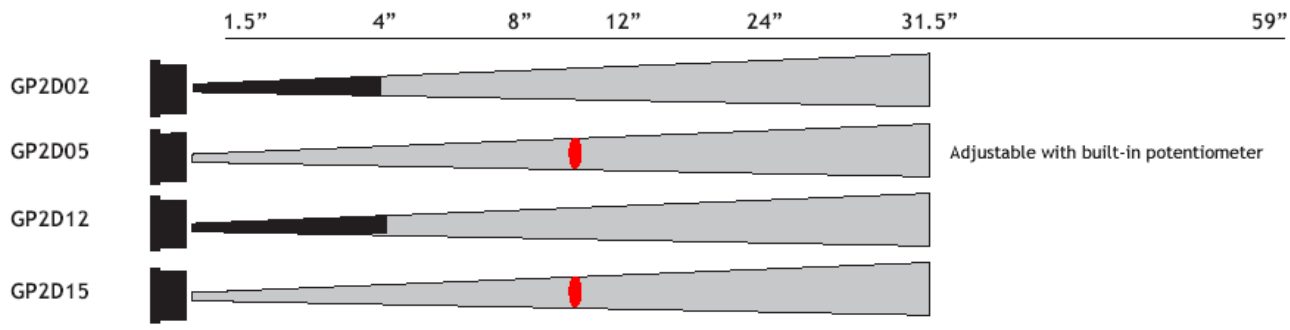


Figure 4.7 – Range Sensor Chart

Sample code for the rotation of the intelliBrain bot

```
private synchronized void doRotate() {

        Pose pose = mLocalizer.getPose();
        float error = mTargetHeading - pose.heading;
        // choose the direction of rotation that results
        // in the smallest angle

        if (error > PI)
                error -= TWO_PI;
        else if (error < -PI)
                error += TWO_PI;
        float absError = (error >= 0.0f) ? error : -error;

        if (absError < mRotateThreshold) {
                mLeftMotor.setPower(Motor.STOP);
                mRightMotor.setPower(Motor.STOP);
                mState = STOP;
                // notify listener the operation is complete
                updateListener(true, null);
                // signal waiting thread we are at the destination
                notify();
        }
        else if (error > 0.0f) {
                mLeftMotor.setPower(-mRotatePower);
                mRightMotor.setPower(mRotatePower);
                }
        else {
                mLeftMotor.setPower(mRotatePower);
                mRightMotor.setPower(-mRotatePower);
        }
}


private synchronized void moveTo(float x, float y, boolean wait,
                                        NavigatorListener listener) {
                updateListener(false, listener);
                mDestinationX = x;
                mDestinationY = y;
                mState = MOVE_TO;

                if (wait) {
                        try {
```

```
                                wait();
                                }
catch (InterruptedException e) {}
                        }
}
```

5. Testing

5.1 Testing IntelliBrain bot

The two-shaft encoders work on a simple principle of sensing low or high voltage by sending an infrared signal toward the wheels. There are 8 holes in each wheel and by sensing each edge of the wheel the counter will add up to 16 when a wheel makes one full rotation.

Multiplying this by the diameter of the wheel we get the distance traveled if the robot is moving straight forward or backwards.

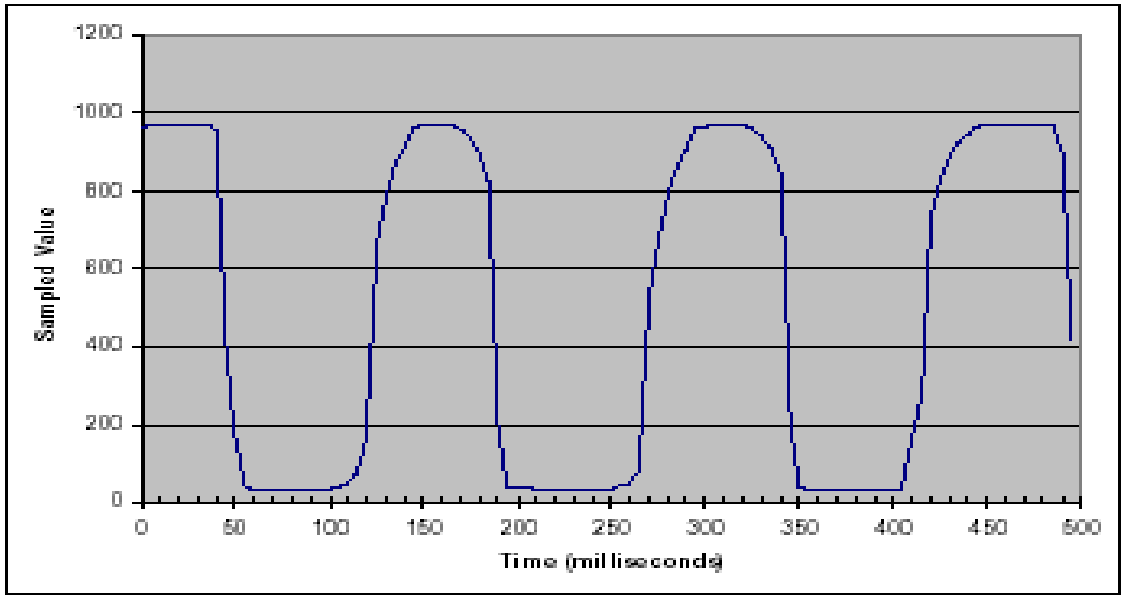


Figure 4.8 – Encoder Run Graph

When the wheels are moving forward constantly we can see that the voltage goes from high to low constantly and we can represent it by graphing the voltages for every millisecond. We can see from the graph that from the high-to-high point on the graph it takes about 150 milliseconds. This is the approximate time it takes for one hole and one spoke to pass the sensor, if multiply it by 8 (there are 8 holes in the wheel) we see that it takes approximately 1.2 seconds for one full rotation of the wheel. And from there we can estimate that the top speed of the robot is 50 rotations per minute if servos are at full speed.

During the testing we have encountered numerous problems with the sensors, such as the inaccuracy when exposed to direct sunlight and inaccurate distance reading when the object is too close.

A) Module Tested

The IntelliBrain bot was tested to move to four x and y coordinates that made up a square with sides of 16 inches.

B) Inputs analyzed

The gain of error was set to 6, 25, 100 in different runs.

C) Expected outputs

The robot was expected to be relatively accurate and close to the x and y coordinates and no further away from the destination than an inch**.**

D) Actual outputs

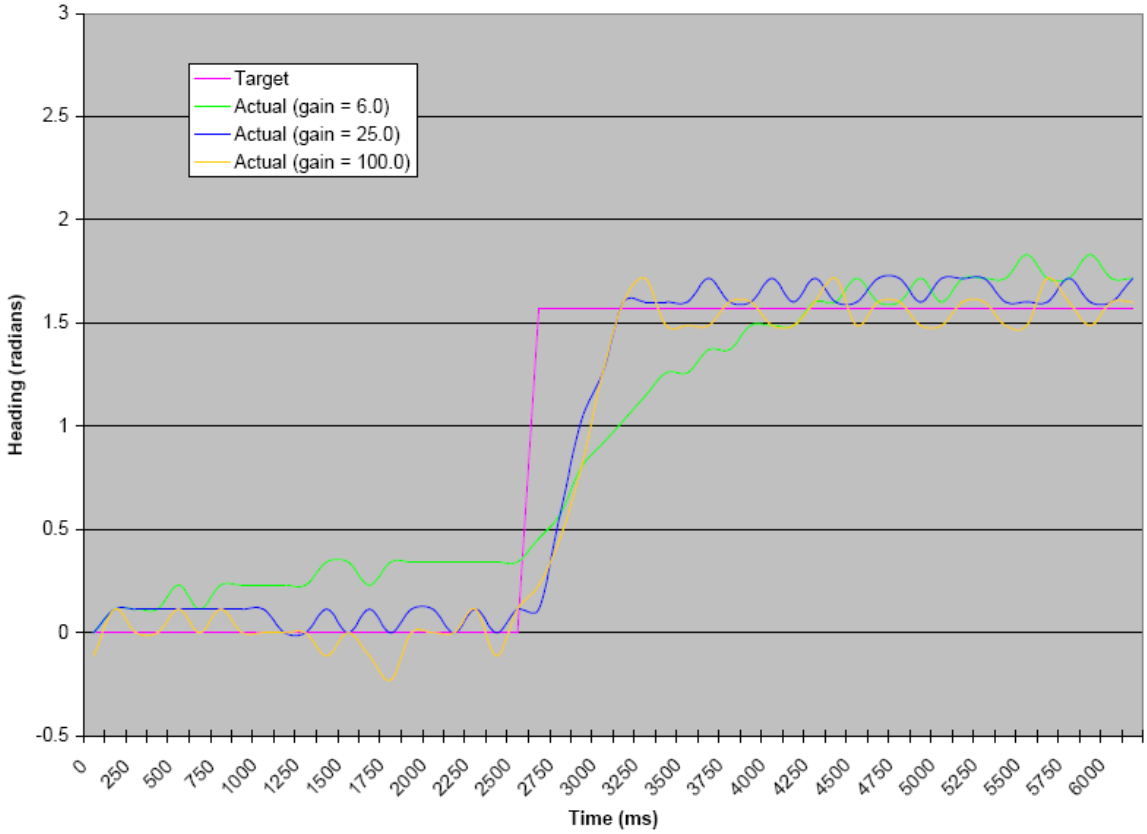


Figure 4.9 – Sample Runs Graph

We can see from the figure 4.9 that the robot was oscilating too widely when the gain was set to a 100 and 6. It was relatevly accurate when the gain was set to 25. It was close to the destination with one inch of error.

5.2 RTX Robot

A) Module Tested

RTX robot was tested to see how accurately and how effectively it moves it's motors to it's maximum and minimum positions.

B) Inputs analyzed

The maximum coordinates for joints were set to

- Column: 0 to -3550
- Shoulder: 2633 to -2630
- Elbow: 2278 to -2645
- Wrist Yaw: 846 to -846
- Wrist 1: -1326 to 0
- Wrist 2: 1195 to -1195
- Gripper: 1200 to 0

C) Expected outputs

The robot was expected to be relatively accurate within millimeters and very close to the set coordinates.

D) Actual outputs

The robot was impressively accurate to the maximum and minimum coordinates we set and was on destination within millimeters. Five sample runs were conducted and the robot produced the same results.

6. Conclusion

To program a robot successfully takes high knowledge of software engineering and object oriented programming. Java's support for multiple threads makes it possible it easy for the robot to run multiple sensors and servos at the same time. Also, Java's great ability to catch exceptions helps a great deal by showing up errors rather than just exiting the program. I can see now why the developers of the robot chose java instead of C/C++. It would be very hard to report certain hardware issues when using C/C++.

We shall see an increase in demand for useful robots from automatic lawnmowers to military scouts. It is a very difficult and challenging but profitable field to get into. No robot is perfect, and it shall most likely never be error free. Sometimes in robotics you have to accept close enough to be perfect. The navigation system is relatively accurate within a couple of inches, so we have to accept that that is good enough. If this was a rover exploring mars we might settle for that. But if it was a mine sweeper, then we might want it to be a little bit more precise.

7. References

[1] Ridgesoft Inc., www.ridgesoft.com

[2] Simulation of mobile robots in virtual environments,

http://alumni.media.mit.edu/~jhe/Publicaciones/CORE2003.pdf

[3] Mobile Robot Simulation with Realistic Error Models, http://robotics.ee.uwa.edu.au

[4] UMI Robot User and Programmer's Manual, for the UMI RT100+ Win32 Library, www.eng.uts.edu.au/~carlo/pdf/Hitsquad_Robot_Manual.pdf

[5] Robot Noughts & Crosses, R.J Smith

[6] C++ library for UMI RTX robot users.telenet.be/emlab/PDFbestanden/RTXproject.pdf

[7] RTX Communications manual, UMI, http://www.staffs.ac.uk/personal/engineering_and_technology/sow1/Robotics/RTX/rtx.htm

8. Appendix

8.1 sample code for IntelliBrain Bot

This code is used for the main navigation of the robot

Sample Code:

```
// this code navigates the robot to x and y coordinates

import com.ridgesoft.robotics.Motor;

public class DifferentialDriveNavigator extends Thread implements Navigator {
        private static final float PI = 3.14159f;
        private static final float TWO_PI = PI * 2.0f;
        private static final float PI_OVER_2 = PI / 2.0f;
        private static final int STOP = 0;
        private static final int GO = 1;
        private static final int MOVE_TO = 2;
        private static final int ROTATE = 3;
        private Motor mLeftMotor;
        private Motor mRightMotor;
        private Localizer mLocalizer;
        private int mDrivePower;
        private int mRotatePower;
        private int mPeriod;
        private int mState;
        private float mDestinationX;
        private float mDestinationY;
        private float mTargetHeading;
        private float mGain;
        private float mGoToThreshold;
        private float mRotateThreshold;
        private NavigatorListener mListener;

public DifferentialDriveNavigator(
```

```
                        Motor leftMotor, Motor rightMotor,
                        Localizer localizer,
                        int drivePower, int rotatePower,
                        float gain,
                        float goToThreshold, float rotateThreshold,
                        int threadPriority, int period) {

        mLeftMotor = leftMotor;
        mRightMotor = rightMotor;
        mLocalizer = localizer;
        mDrivePower = drivePower;
        mRotatePower = rotatePower;
        mGain = gain;
        mGoToThreshold = goToThreshold;
        mRotateThreshold = rotateThreshold;
        mPeriod = period;
        mState = STOP;
        mListener = null;
        setPriority(threadPriority);
        setDaemon(true);
        start();
        }

private void updateListener(boolean completed,
                                NavigatorListener newListener) {
                        if (mListener != null)
                           mListener.navigationOperationTerminated(completed);
                           mListener = newListener;
                          }

private float normalizeAngle(float angle) {

        while (angle < -PI)
                        angle += TWO_PI;
        while (angle > PI)
                        angle -= TWO_PI;
        return angle;
}

private synchronized void goHeading() {
        Pose pose = mLocalizer.getPose();
        float error = mTargetHeading - pose.heading;
                        if (error > PI)
                                error -= TWO_PI;
                        else if (error < -PI)
                                error += TWO_PI;
        int differential = (int)(mGain * error + 0.5f);
        mLeftMotor.setPower(mDrivePower - differential);
        mRightMotor.setPower(mDrivePower + differential);
        }

private synchronized void goToPoint() {

        Pose pose = mLocalizer.getPose();
        float xError = mDestinationX - pose.x;
        float yError = mDestinationY - pose.y;
        float absXError = (xError > 0.0f) ? xError : -xError;
        float absYError = (yError > 0.0f) ? yError : -yError;

                        if ((absXError + absYError) < mGoToThreshold) {
                                // stop
                                mLeftMotor.setPower(Motor.STOP);
                                mRightMotor.setPower(Motor.STOP);
                                mState = STOP;
                                // notify listener the operation is complete
                                updateListener(true, null);
                                // signal waiting thread we are at the destination
                                notify();
                                }
          else {
                        // adjust heading and go that way
                        mTargetHeading = (float)Math.atan2(yError, xError);
                        goHeading();
                        }
        }

private synchronized void doRotate() {

        Pose pose = mLocalizer.getPose();
        float error = mTargetHeading - pose.heading;
```

```
                // choose the direction of rotation that results
                // in the smallest angle

                if (error > PI)
                        error -= TWO_PI;
                else if (error < -PI)
                        error += TWO_PI;
                float absError = (error >= 0.0f) ? error : -error;

                if (absError < mRotateThreshold) {
                        mLeftMotor.setPower(Motor.STOP);
                        mRightMotor.setPower(Motor.STOP);
                        mState = STOP;
                        // notify listener the operation is complete
                        updateListener(true, null);
                        // signal waiting thread we are at the destination
                        notify();
                }
                else if (error > 0.0f) {
                        mLeftMotor.setPower(-mRotatePower);
                        mRightMotor.setPower(mRotatePower);
                        }
                else {
                        mLeftMotor.setPower(mRotatePower);
                        mRightMotor.setPower(-mRotatePower);
                }
}


private synchronized void moveTo(float x, float y, boolean wait,
                                                NavigatorListener listener) {
                        updateListener(false, listener);
                        mDestinationX = x;
                        mDestinationY = y;
                        mState = MOVE_TO;

                        if (wait) {
                                try {
                                        wait();
                                        }
                                catch (InterruptedException e) {}
                                }
                        }


public void moveTo(float x, float y, boolean wait) {
                                moveTo(x, y, wait, null);
                                }

public void moveTo(float x, float y,
                                        NavigatorListener listener) {
                                moveTo(x, y, false, listener);
                                                }

public synchronized void turnTo(float heading, boolean wait,
                                NavigatorListener listener) {
                                updateListener(false, listener);
                                mTargetHeading = normalizeAngle(heading);
                                mState = ROTATE;

                        if (wait) {
                                try {
                                        wait();
                                        }
                                catch (InterruptedException e) {}
                                }
                        }

public void turnTo(float heading, boolean wait) {
                        turnTo(heading, wait, null);
                        }


public void turnTo(float heading, NavigatorListener listener) {
                                turnTo(heading, false, listener);
                                        }

public synchronized void go(float heading) {
                        mTargetHeading = normalizeAngle(heading);
                        mState = GO;
```

```
                updateListener(false, null);
                }

public synchronized void stop() {
        mLeftMotor.setPower(Motor.STOP);
        mRightMotor.setPower(Motor.STOP);
        mState = STOP;
        updateListener(false, null);
        }

        public void run() {
                try {
                        while (true) {
                                switch (mState) {
                                        case MOVE_TO:
                                         goToPoint();
                                        break;
                                   case GO:
                                        goHeading();
                                        break;
                                        case ROTATE:
                                        doRotate();
                                        break;
                                        default: // stopped
                                        break;
                                        }
                        Thread.sleep(mPeriod);
                        }
                }
                catch (Throwable t) {
                        t.printStackTrace();
                                }
                }
}
```

8.2 sample code for RTX robot

This code is for the main graphical user interface controller.

```
package interRob;

// Written by: Jonathan Eren and Neven Skoro

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.border.Border;


public class Melvin_GUI implements ActionListener
{
          boolean inputOk;

             String[] degString;
             float[] nextPos;
             float[] currentPos;

          Float f;
         // creates the button objects
         JButton ConnectBTN = new JButton("Connect");
         JButton ONBTN = new JButton("ON");
         JButton OFFBTN = new JButton("OFF");
         JButton HomeBTN = new JButton("Home");
         JButton MoveBTN = new JButton("Move");
         JButton DSBTN = new JButton("Stop");
         JButton ExitBTN = new JButton("Exit");

         JButton Zed_MoveBTN = new JButton("Move");
         JButton Sh_MoveBTN = new JButton("Move");
         JButton Elbow_MoveBTN = new JButton("Move");
         JButton WristYaw_MoveBTN = new JButton("Move");
         JButton W1_MoveBTN = new JButton("Move");
         JButton W2_MoveBTN = new JButton("Move");
         JButton Gripper_MoveBTN = new JButton("Move");
```

```
      // create text field objects
      JTextField Pos_Zed = new JTextField("-375");
      JTextField Pos_Sh = new JTextField("0");
      JTextField Pos_El = new JTextField("0");
      JTextField Pos_WY = new JTextField("0");
      JTextField Pos_W1 = new JTextField("0");
      JTextField Pos_W2 = new JTextField("0");
      JTextField Pos_GR = new JTextField("0");

      // create int values for position of robot
      int _Zed, _Sh, _El, _Wy, _W1, _W2, _Gr;
      ComCon comCon;
      JPanel p = new JPanel();
      JTextArea Test = new JTextArea();
      Border loweredbevel = BorderFactory.createLoweredBevelBorder();
      Border raisedbevel = BorderFactory.createRaisedBevelBorder();
      Border compound = BorderFactory.createCompoundBorder( raisedbevel, loweredbevel);

      JFrame F = new JFrame();

public Melvin_GUI()
{

  inputOk = true;

  degString = new String[7];
  nextPos = new float[7];
  currentPos = new float[7];
        // set layout to the frame itself to borderlayout
   F.setLayout(new BorderLayout());

   // edit output area to get desired result of a
   // non editable scrolling textarea
   Test.setEditable( false );
   Test.setLineWrap( true );
   Test.setWrapStyleWord( true );
   // JScrollPane allows the JTextArea to be scrolling
   // otherwise it would just have text off screen and not visible
   JScrollPane scrollingResult = new JScrollPane(Test);

   // register the 2 buttons with actionListener events
   ConnectBTN.addActionListener(this);
   ONBTN.addActionListener(this);
   OFFBTN.addActionListener(this);
   HomeBTN.addActionListener(this);
   MoveBTN.addActionListener(this);
   DSBTN.addActionListener(this);
   ExitBTN.addActionListener(this);

   Zed_MoveBTN.addActionListener(this);
   Sh_MoveBTN.addActionListener(this);
   Elbow_MoveBTN.addActionListener(this);
   WristYaw_MoveBTN.addActionListener(this);
   W1_MoveBTN.addActionListener(this);
   W2_MoveBTN.addActionListener(this);
   Gripper_MoveBTN.addActionListener(this);

   // change color of the buttons
   ConnectBTN.setBackground(Color.ORANGE );
   ONBTN.setBackground(Color.ORANGE );
   OFFBTN.setBackground(Color.ORANGE );
   HomeBTN.setBackground(Color.ORANGE );
   MoveBTN.setBackground(Color.ORANGE );
   DSBTN.setBackground(Color.ORANGE );
   ExitBTN.setBackground(Color.ORANGE );
   Zed_MoveBTN.setBackground(Color.ORANGE );
   Sh_MoveBTN.setBackground(Color.ORANGE );
   Elbow_MoveBTN.setBackground(Color.ORANGE );
   WristYaw_MoveBTN.setBackground(Color.ORANGE );
   W1_MoveBTN.setBackground(Color.ORANGE );
   W2_MoveBTN.setBackground(Color.ORANGE );
   Gripper_MoveBTN.setBackground(Color.ORANGE );

   // east frame
   p = new JPanel();
   F.add("East", p);

   // north frame
   p = new JPanel();
   p.setLayout(new GridLayout(1,5));
   p.add(new Label(" "));
```

```
   p.add(MoveBTN);
   p.add(DSBTN);
   p.add(ExitBTN);
   p.add(new Label(" "));
   F.add("North", p);

   // west frame
   p = new JPanel();
   p.setLayout(new GridLayout( 8, 1));
   p.add(ConnectBTN);
   p.add(new Label(" "));
   p.add(ONBTN);
   p.add(new Label(" "));
   p.add(OFFBTN);
   p.add(new Label(" "));
   p.add(HomeBTN);
   p.add(new Label(" "));
   F.add("West", p);

   // South frame
   p = new JPanel();
   p.setLayout( new GridLayout( 3, 7));
   p.setBorder( raisedbevel );
   p.add(new Label("ZED"));
   p.add(new Label("Shoulder"));
   p.add(new Label("Elbow"));
   p.add(new Label("Wrist Yaw"));
   p.add(new Label("Wrist 1"));
   p.add(new Label("Wrist 2"));
   p.add(new Label("Gripper"));
   p.add(Pos_Zed);
   p.add(Pos_Sh);
   p.add(Pos_El);
   p.add(Pos_WY);
   p.add(Pos_W1);
   p.add(Pos_W2);
   p.add(Pos_GR);
   p.add(Zed_MoveBTN);
   p.add(Sh_MoveBTN);
   p.add(Elbow_MoveBTN);
   p.add(WristYaw_MoveBTN);
   p.add(W1_MoveBTN);
   p.add(W2_MoveBTN);
   p.add(Gripper_MoveBTN);
   F.add("South", p);

   // Center frame
   p = new JPanel();
   p.setLayout( new GridLayout( 1, 0));
   p.setBorder(compound);
   p.add(scrollingResult);
   F.add("Center", p);

   Test.append("Welcome to the R.T.X. Control Program." + '\n');
   Test.append("Written by: Jonathan Eren and Neven Skoro" + '\n');
   Test.append("Press the connect button to start a connection with the COM port" + '\n');
   Test.append("Press ON to get started!" + '\n' + '\n');
   Test.append("<-------------------------------------------------------------->" + '\n' + '\n');

   // resizes screen to desired result
   F.resize(600, 300);
   F.setVisible( true );
}

// put code to call the classes to move the robot in here
// begin method actionPerformed
public void actionPerformed(ActionEvent e)
{
         if( e.getSource() == ConnectBTN)
         {
                   Test.append("Connecting..." + '\n');
              comCon = new ComCon();

         }
         else if( e.getSource() == ONBTN)
         {
             Test.append("Turning ON" + '\n');
                       InterRob.on();
                  Test.append("Robot is ON" + '\n');

         }
```

```
else if( e.getSource() == OFFBTN)
{
 Test.append("Turning OFF" + '\n');
        InterRob.stop();

}
else if( e.getSource() == HomeBTN)
{
    Test.append("Going Home" + '\n');
        InterRob.cal();
}
else if( e.getSource() == DSBTN)
{
        InterRob.stop();
        Test.append("DEAD STOP" + '\n');
}
else if( e.getSource() == ExitBTN)
{
        System.exit(0);
}
try
{
        if( e.getSource() == Zed_MoveBTN)
        {
                _Zed = Integer.valueOf( Pos_Zed.getText()).intValue();
                Test.append("Moving Zed to location: " + _Zed  + '\n');
                inputOk=true;
        checkPosition();
         if(inputOk){

         InterRob.pushPos(
         InterRob.column.degToInt(_Zed),
         InterRob.shoulder.degToInt(0),
         InterRob.ellbow.degToInt(0),
         InterRob.wristH.degToInt(0),
         InterRob.wristV.degToInt(0),
         InterRob.wristT.degToInt(0),
         InterRob.gripper.degToInt(0));
         }
         }
         else if( e.getSource() == Sh_MoveBTN)
         {
                 _Sh= Integer.valueOf( Pos_Sh.getText()).intValue();
                 Test.append("Moving Shoulder to location: " + _Sh  + '\n');
                 checkPosition();
                 if(inputOk){

         InterRob.pushPos(
         InterRob.column.degToInt(0),
         InterRob.shoulder.degToInt(_Sh),
         InterRob.ellbow.degToInt(0),
         InterRob.wristH.degToInt(0),
         InterRob.wristV.degToInt(0),
         InterRob.wristT.degToInt(0),
         InterRob.gripper.degToInt(0));
         }
         }
         else if( e.getSource() == Elbow_MoveBTN)
         {
                 _El= Integer.valueOf( Pos_El.getText()).intValue();
                 Test.append("Moving Elbow to location: " + _El + '\n' );
                 checkPosition();
                 if(inputOk){

         InterRob.pushPos(
         InterRob.column.degToInt(0),
         InterRob.shoulder.degToInt(0),
         InterRob.ellbow.degToInt(_El),
         InterRob.wristH.degToInt(0),
         InterRob.wristV.degToInt(0),
         InterRob.wristT.degToInt(0),
         InterRob.gripper.degToInt(0));
         }
         }
         else if( e.getSource() == WristYaw_MoveBTN)
         {
                 _Wy = Integer.valueOf( Pos_WY.getText()).intValue();
                 Test.append("Moving Wrist Yaw to location: " + _Wy + '\n');
                 checkPosition();
                 if(inputOk){
```

```
 InterRob.pushPos(
 InterRob.column.degToInt(0),
 InterRob.shoulder.degToInt(0),
 InterRob.ellbow.degToInt(0),
 InterRob.wristH.degToInt(_Wy),
 InterRob.wristV.degToInt(0),
 InterRob.wristT.degToInt(0),
 InterRob.gripper.degToInt(0));
 }

 }
 else if( e.getSource() == W1_MoveBTN)
 {
         _W1 = Integer.valueOf( Pos_W1.getText()).intValue();
         Test.append("Moving Wrist 1 to location: " + _W1 + '\n' );
         checkPosition();
         if(inputOk){

 InterRob.pushPos(
 InterRob.column.degToInt(0),
 InterRob.shoulder.degToInt(0),
 InterRob.ellbow.degToInt(0),
 InterRob.wristH.degToInt(0),
 InterRob.wristV.degToInt(_W1),
 InterRob.wristT.degToInt(0),
 InterRob.gripper.degToInt(0));

 }
 }
 else if( e.getSource() == W2_MoveBTN)
 {
         _W2 = Integer.valueOf( Pos_W2.getText()).intValue();
         Test.append("Moving Wrist 2 to location: " + _W2 + '\n' );
         checkPosition();
         if(inputOk){

 InterRob.pushPos(
 InterRob.column.degToInt(0),
 InterRob.shoulder.degToInt(0),
 InterRob.ellbow.degToInt(0),
 InterRob.wristH.degToInt(0),
 InterRob.wristV.degToInt(0),
 InterRob.wristT.degToInt(_W2),
 InterRob.gripper.degToInt(0));

 }
 }
 else if( e.getSource() == Gripper_MoveBTN)
 {
         _Gr = Integer.valueOf( Pos_GR.getText()).intValue();
         Test.append("Moving Gripper to location: " + _Gr + '\n' );
         checkPosition();
         if(inputOk){

 InterRob.pushPos(
 InterRob.column.degToInt(0),
 InterRob.shoulder.degToInt(0),
 InterRob.ellbow.degToInt(0),
 InterRob.wristH.degToInt(0),
 InterRob.wristV.degToInt(0),
 InterRob.wristT.degToInt(0),
 InterRob.gripper.degToInt(_Gr));

 }
 }
 else if( e.getSource() == MoveBTN)
 {
         _Zed = Integer.valueOf( Pos_Zed.getText()).intValue();
         _Sh= Integer.valueOf( Pos_Sh.getText()).intValue();
         _El= Integer.valueOf( Pos_El.getText()).intValue();
         _Wy = Integer.valueOf( Pos_WY.getText()).intValue();
         _W1 = Integer.valueOf( Pos_W1.getText()).intValue();
         _W2 = Integer.valueOf( Pos_W2.getText()).intValue();
         _Gr = Integer.valueOf( Pos_GR.getText()).intValue();

         inputOk=true;
         checkPosition();
if(inputOk){

 InterRob.pushPos(
 _Zed,_Sh,_El,_Wy,_W1,_W2,_Gr);
```

```
                    }
                            Test.append("Moving all motors to location:" + '\n');
                            Test.append("-->Zed: " + _Zed + '\n');
                            Test.append("-->Shoulder: " + _Sh + '\n');
                            Test.append("-->Elbow: " + _El + '\n');
                            Test.append("-->Wrist Yaw: " + _Wy + '\n');
                            Test.append("-->Wrist 1: " + _W1 + '\n');
                            Test.append("-->Wrist 2: " + _W2 + '\n');
                            Test.append("-->Gripper: " + _Gr + '\n');
                    }
            }
            catch(NumberFormatException E1)
            {
                    Test.append("Error: Please Recheck your input!" + '\n');
            }

          }// end method actionPerformed

   public static void main(String args[])
   {
          Melvin_GUI G1 = new Melvin_GUI();
   }

   public void checkPosition() {
            if(_Zed < InterRob.column.maxCoord | _Zed>0){
           inputOk=false;
           System.out.println("c");
       }
       if(_Sh > InterRob.shoulder.maxDeg | _Sh<-InterRob.shoulder.maxDeg){
           inputOk=false;
           System.out.println("s");
       }
       if(_El > InterRob.ellbow.maxDeg | _El<-InterRob.ellbow.maxDeg){
           inputOk=false;
           System.out.println("e");
       }
       if(_Wy > InterRob.wristH.maxDeg | _Wy<-InterRob.wristH.maxDeg){
           inputOk=false;
           System.out.println("wh");
       }
       if(_W1 <-InterRob.wristV.maxDeg | _W1>0){
           inputOk=false;
           System.out.println("wv");
       }
       if(_W2 > InterRob.wristT.maxDeg | _W2<-InterRob.wristT.maxDeg){
           inputOk=false;
           System.out.println("wt");
       }
       if(_Gr > InterRob.gripper.maxCoord | _Gr<0){
           inputOk=false;
           System.out.println("g");
       }
   }
}
```