Algorithm for Medical Nanobots using C++

Manu Mitra
Electrical Engineering Department, University of Bridgeport, Bridgeport, CT, USA

Abstract
Medical nanobots not only repair cells and tissues but also multiple nanobots can help cure various types of diseases such as cancers, infection or to remove infected cells/tissues. To automate Medical Nanobot we need program to detect it and work on it; and there may be the need for manual work to move Medical Nanobot and perform operations.

A very basic software attempt is made for Medical Nanobot using C/C++ later same methodology can be used for advanced programming of Medical Nanobot. In this paper flow diagram of medical nanobot for disease detection, removal of infected cells, tissues, repairing the cells, tissues and continuous monitoring is made including various pseudo code is demonstrated such as setting up, driving nanobots for manual and automatic, auto pan/tilt, nano gyro sensors for disease detection, camera configuration, nano servo mechanism, handling interrupts and synchronization of nanobot using C++.

Keywords
Nanobot; Nano Robot; Medical Nanobot; Nano Machines; Cell Repair; Tissue Repair; Algorithm; C++

Introduction
Medical uses of nano devices incorporate plaque evacuation and heart repair. They should relocate to a foreordained site and stay in that area to finish the task.

Multiple medical nanobots can be used collectively for medical applications to map the human body, to regulate the cardio-vascular system, for insulin regulation, for targeted drug delivery, for diagnosis of cellular pathologies and for destroying tumor cells [1].

Another necessity of nano machine is that it works autonomously, free of outside control. Physical, electrical, and compound responses can deliver a reaction yet the presentation of these stimuli victimizes the device of the benefit of autonomous task and they additionally can create a reaction in the nanobots encompassing condition.

An innovative theory in the utilization of these nano devices to battle disease that includes utilizing silicon nanomachines with a thin covering of gold and light in the close infrared range.

Light in the 700-1000 nanometer range will go through tissue with insignificant ingestion. At the point when this close infrared light strikes this specific sort of nanomedibot, the device gets hot because of the wavering of the metal’s electrons in response to the light. Utilizing MRI to definitely put the nanomedibots in the dangerous district, the light makes the devices warmth to 131 degrees Fahrenheit which wrecks the destructive cells yet doesn't harm encompassing tissues.

Likewise with respect to disease treatment, ribonucleic corrosive obstruction is a technique that assaults tumors on a hereditary level. Nanobots weighed down with meddling RNA that deactivates the protein creation of the growth and murders the danger would connect themselves to the tumor and convey the deadly hereditary material. In addition of expelling plaque from blood vessel dividers; they could likewise be utilized to discover arterial weakness.

Nanobots may likewise be utilized to distinguish particular chemicals or poisons and could give early cautioning of organ disappointment or tissue dismissal. Additionally they can be used to take biometric estimations, they might be utilized to screen the general soundness of a person.

These nano devices may discover application in an assortment of mechanical applications. Research is continuous into utilizing them in the oil business.

In addition, current research is examining the ir application in nano photonics to create light more effectively. PC circuits might be delivered by these small devices. They could make circuits on a very smaller scale than current drawing systems and would take into consideration to manufacture of extremely small processors and chips [2-4].

One of the major advantage of nanobots it can be considered as a way of delivering differentiated stem cells to various positions in the body. Stem cell research has been a...
huge increase in regenerative medicine. Nanobots help to enhance its impact on medicine in the near future by providing an effective way of delivering them [5][Figure 1][6].

Flow Diagram of Medical Nanobot

A. Creating nanobot base class

Pseudocode for simple nanobot base class Example I

Below is the example code to create a nanobot's base class [7].

```c
1. class SimpleNanobot: public NanobotBase {
2. public: SimpleRobot(void);virtual void Autonomous();
3. virtual void StartCompetition();virtual void StartRobotMain();virtual void StartCompetition();
4. private: bool m_nanobotMainOverridden;
5. }
```

Pseudocode for Start Competition method Example II

```c
1. void SimpleNanobot::StartCompetition()
2. {
3.   while (IsDisabled()) Wait(0.01); // waiting for match to start
4.   if (IsAutonomous()) // making nanobot autonomous
5.   {
6.     Autonomous(); // to run user provided Autonomous code
7.   }
8.   while (IsAutonomous()) Wait(0.01); // nanobot wait until end of autonomous
9.   while (IsDisabled()) Wait(0.01); // make sure that nanobot is enabled
10.  OperatorControl(); // start user provided OperatorControl
11. }
```

B. Manual and Autonomous Pseudocodes for Nanobots

An attempt is made for basic manual execution of program for a Medical Nanobot using C/C++ language.

Basic program of nanobot using C

Below is the pseudocode in C program that demonstrates driving the Nanobot for 2 seconds forward in Autonomous and in arcade mode for the Operator Control.

```c
1. #include <Libraryfile.h> // Library File for Nanobot
2. #include "Nanobot.h" // Includes all functions, recursion loops for Nanobot
3. static const UINT32 LEFT_Nanobot_PORT = 1; // Left Navigation for Nanobot
4. static const UINT32 RIGHT_Nanobot_PORT = 2; // Right Navigation for Nanobot
5. static const UINT32 JOYSTICK_PORT = 1; // Manual movement if automation is failed
6. void Initialize(void) {
7.   CreateRobotDrive(LEFT_Nanobot_PORT, RIGHT_Nanobot_PORT);
8.   SetWatchdogExpiration(0.1); // Watchdog timer
9. }
10. void Autonomous(void) {
11.   SetWatchdogEnabled(false);
12.   Drive(0.5, 0.0);
13.   Wait(2.0);
14.   Drive(0.0, 0.0);
15. }
16. void OperatorControl(void) {
17.   SetWatchdogEnabled(true);
18.   while (IsOperatorControl()) {
19.     WatchdogFeed();
20.     ArcadeDrive(JOYSTICK_PORT);
21.   }
22. }
23. START_ROBOT_CLASS(Nanobot);
```

Simple pseudocode for nanobot Pan/Tilt in C++

Below is the example code for Pan/Tilt in C++ [8].

```c
1. #include <Libraryfile.h> // This loads the Nanobot script, allowing you to use specific functions below
2. #include <Nanobot.h> // Includes all functions, recursion loops for Nanobot
3. Nanobot myNanobot; // create Nanobot object to control a Nanobot
4. int pos = 0; // variable to store the Nanobot position
5. void setup() // required in all Nanobot Software code
6. {
7.   myNanobot.attach(9); // attaches the Nanobot on pin 9 to the Nanobot object
8. }
9. void loop() // required in all Nanobot Software code
10. {
11.   for (pos = 0; pos < 180; pos += 1) // variable ‘pos’ goes from 0 degrees to 180 degrees in steps of 1 degree
12.   {
13.     myNanobot.write(pos); // tell Nanobot to go to
```

Figure 1: Depicts nanobot repairing a damaged DNA (Deoxyribonucleic Acid)
C. Nano Gyro Sensors for Nanobots

Gyro sensors are the angular velocity applied to a vibrating element, the accuracy with which angular velocity is measured differs significantly depending on element material and structural differences. Various characteristics of gyro sensor include but not limited to scale factor, temperature-frequency coefficient, compact size, shock resistance, stability, and noise characteristics etc., [9].

In this program robot drives in a straight line using gyro sensor with nanobotDrive class. The NanobotDrive.Drive method takes the speed and turn rate as arguments; where both vary combination with nanobotDrive class. The NanobotDrive.Drive method.

Pseudocode for nanobot gyro sensor

```cpp
from its initial heading [10].

var angle = gyro.GetAngle();
// get heading

myNanobot.Drive(-1.0, -angle * Kp) // turn to correct heading for nanobot

Wait(0.004);

myNanobot.Drive(0.0, 0.0); // stop Nanobot
```

D. Nano Camera for Nanobots

A nano camera is very important to nanobots. C++ provides initialization, control and image acquisition functionality. StartCameraTask() initializes the camera.

Pseudocode for simple camera initialization

```cpp
if (StartCameraTask() == -1) {

dprintf(LOG_ERROR, “Failed to spawn camera task; Error code %s”, GetErrorText(GetLastError()));
}
```

Pseudocode for camera configuration

```cpp
int frameRate = 15; // valid values 0 - 30
int compression = 0; // valid values 0 - 100

ImageSize resolution = k160x120; // k160x120, k320x240, k640x480

ImageRotation rot = ROT_180; // ROT_0, ROT_90, ROT_180, ROT_270

StartCameraTask(frameRate, compression, resolution, rot); // Nano camera configuration for nanobot
```

Image acquisition through camera

```cpp
double timestamp; // timestamp of image returned

Image * cameraImage = frcCreateImage(IMAQ_IMAGE_HSL);
```

Pseudocode for Color Tracking for Nanobot Example I

```cpp
if (!cameraImage) {
    printf(“error: % s”, GetErrorText(GetLastError()));
}

if (!GetImage(cameraImage, & timestamp)) {
    printf(“error: % s”, GetErrorText(GetLastError()));
}
```

E. Vision and Image Processing for Nanobots

Nanobot’s vision system has to make distinction between objects and in most of all cases it has to track. It is used to automate the process and object detection [11].

Pseudocode for Color Tracking for Nanobot Example I

```cpp
TrackingThreshold tdata = GetTrackingData(BLUE, FLUORESCENT);

ParticleAnalysisReport par;
```

Pseudocode for using specified ranges example II

```cpp
Range hue, sat, lum;
```

```cpp
// hue, sat, lum;
```

```cpp
hue.minValue = 140; // Hue
```

```cpp
hue.maxValue = 155;
```

```cpp
sat.minValue = 100; // Saturation
```

```cpp
sat.maxValue = 255;
```

```cpp
lum.minValue = 40; // Luminance
```

```cpp
lum.maxValue = 255;
```

```cpp
FindColor(IMAQ_HSL, & hue, & sat, & lum, & par); // Specifying range for nanobot
```
Pseudocode for Declaration Class Example III
1. RobotDrive * Nanobot Range greenHue, greenSat, greenLum; // Declaration
   Class for nanobot

Pseudocode for Initialization of nano camera Example IV
1. if (StartCameraTask() == -1) {
   2.   print("Failed to spawn camera task; Error code %s", GetErrorMessage());
   3. }
4. nanobot = new RobotDrive(1, 2); // values for tracking a target - may need tweaking
   in the environment greenHue.minValue = 65; greenHue.maxValue = 00; greenSat.minValue
   = 100; greenSat.maxValue = 255; greenLum .minValue = 100; greenLum .maxValue = 255;

Pseudocode for Automating for nanobot Example IV
1. while (IsAutonomous()) {
   2.   if (FindColor(IMAQ_HSL, & greenHue,
      & greenSat, & greenLum, & par) && par.particleToImagePercent < MAX_PARTICLE_TO_IMAGE_PERCENT
      && par.particleToImagePercent > MIN_PARTICLE_TO_IMAGE_PERCENT) {
      3.       nanobot - > Drive(1.0, (float) par.center_mass_x_normalized);
   4. } else nanobot - > Drive(0.0, 0.0);
   5.   Wait(0.05);
   6. }
7. nanobot - > Drive(0.0, 0.0); // Automating nanobot

F. Nano Servo Mechanism for Nanobots
Nano Servomechanism is also required for nanobots for rotation.

Pseudocode for Nano servo mechanism Example I
1. Servo servo(3); // create a servo on PWM port 3 on the first module
2. float servoRange = servo.GetMaxAngle() - servo.GetMinAngle();
3. for (float angle = servo.GetMinAngle(); // step through range of angles
      angle += servoRange / 10.0) {
   4.   servo.SetAngle(angle); // set servo to angle
   5.   Wait(1.0); // wait 1 second
   7. } // Nano servo mechanism for nanobot - I

Pseudocode for Nano servo mechanism Example II
1. #include "Nano_BaeUtilities.h" // Library files for functions for nano servo mechanism
2. panInit(); // optional parameters can adjust pan speed for nanobot
3. bool targetFound = false;
4. while (!targetFound) {
   5.   panForTarget(servo, 0.0); // Start from 1 to +1 // code to identify target for nanobot
6. } // Nano servo mechanism for nanobot - II

F. Nano Solenoid for Nanobots
Solenoids are used as an actuator. Here solenoid can be used to remotely control if automation fails or any other malfunction. It can be used as steering of nanobot [12].

Pseudocode for nano solenoid
1. Nano_Solenoid * s[8];
2. for (int i = 0; i < 8; i++) s[i] = new Solenoid (i + 1);
   // allocate the Nano Solenoid objects
3. for (int i = 0; i < 8; i++) {
   4.   s[i] - > Set(true); // turn them all on
   5. }
6. for (int i = 0; i < 8; i++) {
   7.   s[i] - > Set(false); // turn them each off in turn
   8.   Wait(1.0);
   9. }
10. for (int i = 0; i < 8; i++) {
   11.   s[i] - > Set(true); // turn them back on in turn
   12.   Wait(1.0);
   13.   delete s[i]; // delete the objects
14. }

G. Synchronization of Nanobots
When there are multiple nanobots then synchronization is required. So that multiple nanobots can perform accordingly to complete the task accordingly.

Pseudocode for synchronization example I
1. {
   2.   Synchronized s(semaphore); // access shared code here
   3. } if (condition) return; // more code here
4. } // Synchronization for nanobot

Pseudocode for critical condition example II
1. {
   2.   Synchronized s(semaphore); // access shared code here
   3. } if (condition) return; // more code here
4. } // Synchronization for nanobot

H. Handling of Interrupts in Nanobots

Pseudocode for handling of interrupts example
1. static int interruptCounter = 0; // The interrupt handler that counts number
   of square wave cycles
2. static void tiHandler(tNIRIO_u32 interruptOccurredMask, void * param) {
   3.   interruptCounter++;
4. }
5. void InterruptTestHandler(void) { // create the two digital ports (Output and Input)
   6.   DigitalOutput digOut(CROSS_CONNECT_A_PORT1);
   7.   DigitalInput digIn(CROSS_CONNECT_A_PORT2); // create the counter that will also count square waves
   8.   Counter counter( & digIn); // initialize the digital output to 0
   9.   digOut.Set(0); // start the counter counting at 0
10.   counter.Reset();
11.   counter.Start(); // register and enable the interrupt handler digIn.
   RequestInterrupts(tiHandler);
Appendix

Example UML logic for finite state machine for nanobot
1. namespace FSM;
2. class MgaObject {
3.     String name;
4.     String position;
5. }
6. class Transition {
7.     isA MgaObject; * transition--1..*StateMachine stateMachine;
8.     1..*transition--1..*AssociationStateState associationStateState;
9. }
10. class State {
11.     isA MgaObject;
12.     0..1 dstTransition--1..*AssociationStateState associationStateState; * state--1..*StateMachine stateMachine;
13.     1..*srcTransition--1..*AssociationStateState associationStateState;
14. }
15. class StateMachine {
16.     isA MgaObject; * stateMachine--1..*RootFolder rootFolder;
17. }
18. class RootFolder {
19.     String name;
20.     0..1 - > * RootFolder rootFolders;
21. }
22. class AssociationStateState {};
23. namespace PrimitiveTypes;
24. class String {};
25. class Integer {};

Example UML logic Architectural description design for Nanobot

Figure 2 is the Example of UML logic Architectural description design for Nanobot.

Example UML logic code architectural description for nanobot
1. namespace ArchitecturalDescription;
2. class Architecture {
3.     0..1 - > 1..*ArchitectureDescription describedBy;
4. }
5. class ArchitectureDescription {
6.     0..1 selectedBy--1..*ArchitectureView selects;
7.     0..1 - > 1..*ArchitectureView organizedBy;
8. }
9. class ArchitectureView {
Figure 3: UML Logic diagram access control for medical nanobot.

Example UML diagram for access control design for nanobot

Figure 3 is the example of UML diagram for access control design for nanobot.

Example UML logic for finite state machine for nanobot

1. // UML class diagram in Umlpe representing a system for managing access to facilities
2. namespace access_control; //Ref_Facility_Type
3. class FacilityType {
4.   code;
5.   description {
6.     Menu, Record, Screen
7.   }
8.   key {
9.     code
10. } //Functional_Area
11. class FunctionalArea {
12.   String code;
13.   0..1 parent-- * FunctionalArea child;
14.   description {
15.     Operations, Control
16.   }
17.   key {
18.     code
19.   }
20. }
21. //Facility_Functional_Area
22. association { * FunctionalArea-- * Facility;
23. }
24. class Facility {
25.   Integer id; * - > 0..1 FacilityType;
26.   Integer access_count;
27.   name;
28.   description;
29.   other_details;
30.   key {
31.     id
32.   }
33. }
34. class Role {
35.   code;
36.   role_description {
37.     Dba, ProjectMgr
38.   }
39.   key {
40.     code
41.   }
Examples of Pseudocodes in C++ is presented, UML logic for Medical nanobot is for demonstration and illustrations only and is not tested. Original code may vary and differ based on operations, model and requirements of multiple functionalities and operations.

**Results**

Basic Algorithm for Medical Nanobot with the flow chart specified in Figure 4, UML logic for Finite State Machine, UML logic Architecture Description for Nanobot, UML logic for Access control design and example of C++ is presented. Simplified pseudocode for nanobot is demonstrated such as setting up, driving nanobots for manual and automatic, auto pan/tilt, nano gyro sensors for disease detection, camera configuration, nano servo mechanism, handling interrupts and synchronization of nanobot using C++.

**Conclusion**

Based on the above results, basic algorithm for Medical Nanobot can constructed using C++ as shown in the Figure 2.

Examples of UML for Finite State Machine, Access Control design, Access control logic, Architectural description can be used for designing software for Medical Nanobot using C++ or any other programming language.

**Discussion**

For construction of Nanobots biodegradable poly(lactide-co-glycolide) (PLGA), an FDA approved polymer, can be used to formulate the nanoparticles to form a nanobot [13].

Advances in delivering therapy, reduction of analytical tools, enhanced computational and memory capabilities and developments in inaccessible communications will be integrated allowing for the development of such nanobots [14].

In order to evaluate the effectiveness of technique, known in literature and simulation results showed the effectiveness technique in terms of achievement, that is the destruction of the cancerous cells, and velocity of destruction [15].

**Acknowledgment**

Author would like to thank Prof. Navarun Gupta, Prof. Hassan Bajwa, Prof. Linfeng Zhang and Prof. Hmurcik for their academic support. Author also thanks anonymous reviewers for their comments.
References

9. Seiko E. Gyro sensors - How they work and what's ahead. 2018