Boffin Access

Open Access Journal | Volume: 1.3

Advances in Computer Sciences

ISSN 2517-5718

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 their 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

Article Information

DOI: 10.31021/acs.20181112
Article Type: Research Article
Journal Type: Open Access
Volume: 1 **Issue:** 3
Manuscript ID: ACS-18-012
Publisher: Boffin Access Limited

Received Date: 23 July 2018
Accepted Date: 18 September 2018
Published Date: 21 September 2018

*Corresponding author:

Mitra M
Electrical Engineering Department
University of Bridgeport
Bridgeport, CT, USA
E-mail: mmitra@my.bridgeport.edu

Citation: Mitra M. Algorithm for Medical Nanobots using C++. Adv Comput Sci. 2018;1(3):112

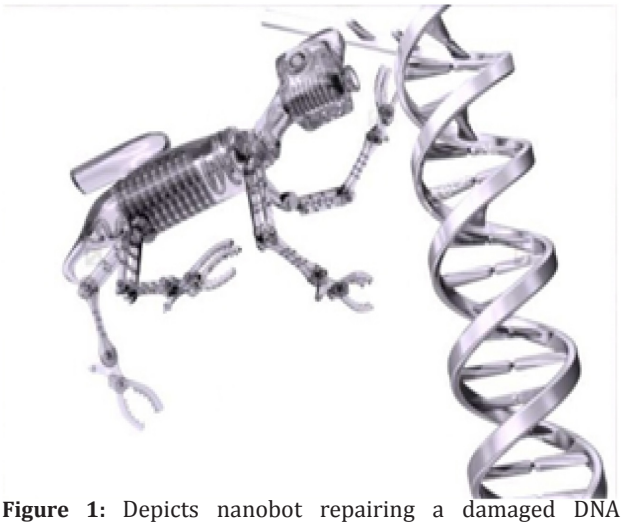

Figure 1: Depicts nanobot repairing a damaged DNA (Deoxyribonucleic Acid)

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].

```
1. class SimpleNanobot: public NanobotBase {
2. public: SimpleRobot(void);virtual void Autonomo
   us(); void OperatorControl();virtual void na
   nobotMain();virtual void StartCompetition()
   ;private: bool m_nanobotMainOverridden;
3. };
```

Pseudocode for Start Competition method Example II

```
1.  void
    SimpleNanobot::StartCompetition
    (void) {

         while (IsDisabled()) Wait(0.01); // waiti
    ng for match to start
2.      if (IsAutonomous()) // making nanobot auto
    nomous
3.      {
4.          Autonomous(); // to run user provided
    Autonomous code
5.      }
6.      while (IsAutonomous()) Wait(0.01); // nano
    bot wait until end of autonomous
7.      while (IsDisabled()) Wait(0.01); // make
    sure that nanobot is enabled
8.      OperatorControl(); // start user provided
    OperatorControl
9.  }
```

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.

In this code Watchdog time class has been used to make sure that the nanobot will stop operating if program does something unexpected or crashes. something unexpected or crashes

```
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].

```
1.  #include < Libraryfile.h > // This loads the
    Nanobot script, allowing you to use specific
    functions below
2.  #include < Nanobot.h > // Includes all func
    tions, 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 degr
    ees in steps of 1 degree
12.         {
13. myNanobot.write(pos); // tell Nanobot to go to
```

```
    position in variable 'pos'
14.             delay(15); // waits 15ms for the
    Nanobot to reach the position
15.         }
16.         for (pos = 180; pos >= 1; pos -= 1) //
    variable 'pos' goes from 180 degrees to 0 degr
    ees
17.         {
18. myNanobot.write(pos); // tell Nanobot to go to
    position in variable 'pos'
19. delay(20); // waits 20ms at each degree
20.         }
21.     }
```

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 combination with nanobotDrive class. The NanobotDrive.Drive method takes the speed and turn rate as arguments; where both vary from -1.0 to 1.0. In this instance uses the gyro returns values that varies either positive or negative degrees as the nanobot's deviates from its initial heading [10].

Simple pseudocode for nanobot gyro sensor

```
1.  class Gyrosensor: public Nanobot {
2.      NanobotDrive myNanobot; // Nanobot
     drive system
3.      Gyro gyro;
4.      static
5.      const float Kp = 0.03;
6.      public: Gyrosensor(void): myNanobot(1,
     2), // initialize the sensors in initial
    ization list
7.          gyro(1)
    {
8.              GetWatchdog().SetExpiration
    (0.1);
9.          }
10.     void Autonomous(void) {

11.         gyro.Reset();
12.         while (IsAutonomous()) {
13.             GetWatchdog().Feed();
14.             float angle = gyro.GetAngle();
    // get heading
15.             myNanobot.Drive(-1.0, -angle
    * Kp)
     // turn to correct heading for nanobot
16.             Wait(0.004);
17.         }
18.         myNanobot.Drive(0.0, 0.0); // stop
    Nanobot
19.     }
20. };
```

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

```
1.  if (StartCameraTask() == -1) {
2.      dprintf(LOG_ERROR, "Failed to
     spawn camera task; Error code %s",
    GetErrorText(GetLastError()));
3.  }  // Nano camera initialization for
     nanobot
```

Pseudocode for camera configuration

```
1.  int frameRate = 15; // valid values 0 - 30
2.  int compression = 0; // valid values 0 – 100
3.  ImageSize resolution = k160x120; // k160x120,
     k320x240, k640480
4.  ImageRotation rot = ROT_180; // ROT_0, ROT_90,
     ROT_180, ROT_270
5.  StartCameraTask(frameRate, compression,
    resolution, rot);  // Nano camera configurati
    on for nanobot
```

Image acquisition through camera

```
1.  double timestamp; // timestamp of image retur
    ned
2.  Image * cameraImage = frcCreateImage(IMAQ_
    IMAGE_HSL);
3.  if (!cameraImage) {
4.      printf("error: % s", GetErrorText(Get
    LastError())
5.      };
6.      if (!GetImage(cameraImage, & timest
    amp)) {
7.          printf("error: % s", GetErrorText
    (GetLastError())
8.          };  // Image acquisition through
    nano camera for nanobot
```

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

```
1.  TrackingThreshold tdata = GetTrackingData(BL
    UE, FLUORESCENT);
2.  ParticleAnalysisReport par;
3.  if (FindColor(IMAQ_HSL, & tdata.hue, & tda
    ta.saturation, & tdata.luminance, & par) {
4.          printf("color found at x = % i, y
    = % i ",  // finding color for nanobot
5.              par.center_mass_x_normalized,
     par.center_mass_y_normalized);
6.          printf("color as percent of image:
     % d ",
7.              par.particleToImagePercent);
8.      }  // Color tracking for nanobot
```

Pseudocode for Using Specified ranges Example II

```
1.  Range hue, sat, lum;
2.  hue.minValue = 140; // Hue
3.  hue.maxValue = 155;
4.  sat.minValue = 100; // Saturation
5.  sat.maxValue = 255;
6.  lum.minValue = 40; // Luminance
7.  lum.maxValue = 255;
8.  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.     printf("Failed to spawn camera
   task; Error code %s", GetErrorText(GetLast
   Error()));
3. }
4. nanobot = new RobotDrive(1, 2); // values
   for tracking a target -may need tweaking
   in the environment greenHue.minValue = 65;
    greenHue.maxValue = 80; 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.particl
   eToImagePercent < MAX_PARTICLE_TO_IMAGE_PERCE
   NT && par.particleToImagePercent > MIN_PARTI
   CLE_TO_IMAGE_PERCENT) {
3.         nanobot - > Drive(1.0, (float) par.cen
   ter_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 Mechanismfor 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(); // ste
   p through range of angles
4. angle < servo.GetMaxAngle(); angle += servoR
   ange / 10.0) {
5. servo.SetAngle(angle); // set servo to angle
6. 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 interr
   uptAssertedMask, void * param) {
3.     interruptCounter++;
4. }
5. void InterruptTestHandler(void) { // c
   reate 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 wil
   l also count square waves
8.     Counter counter( & digIn); // init
   ialize the digital output to 0
9.     digOut.Set(0); // start the counter
    counting at 0
10.     counter.Reset();
11.     counter.Start(); // register and e
   nable the interrupt handler      digIn.
   RequestInterrupts(tiHandler);
```

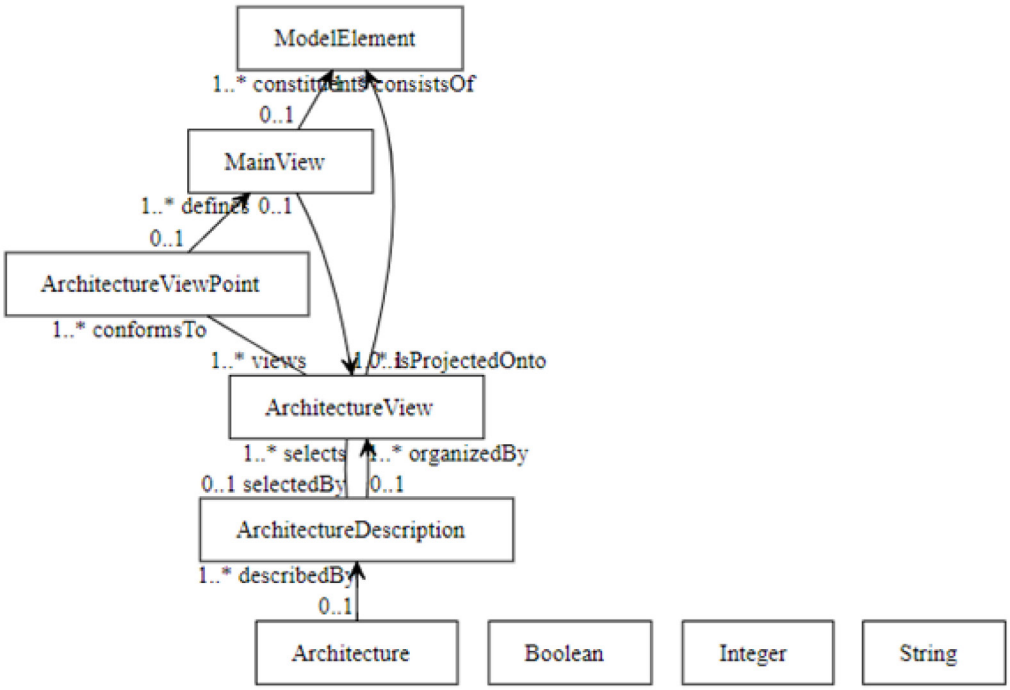

Figure 2: UML Diagram of Architecture Description

```
12.     digIn.EnableInterrupts(); // count
     5 times
13.     while (counter.Get() < 5) {
14.         Wait(1.0);
15.         digOut.Set(1);
16.         Wait(1.0);
17.         digOut.Set(0);
18.     } // verify correct operation
19.     if (interruptCounter == 5 && counter.
    Get() == 5) printf("Test passed!\n"); //
    free resources
20.     digIn.DisableInterrupts();
21.     digIn.CancelInterrupts();
22. }
23. END_TEST(TestInterruptHandler)  //
    Interrupt Handler for Nanobot
```

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..*StateMach
    ine stateMachine;
8.      1..*transition--1..*AssociationStateState
    associationStateState;
9.  }
10. class State {
11.     isA MgaObject;
12.     0..1 dstTransition--1..*AssociationStateS
    tate associationStateStatedst; * state--1..*S
    tateMachine stateMachine;
13.     1..*srcTransition--1..*AssociationStateSta
    te associationStateStatesrc;
14. }
15. class StateMachine {
16.     isA MgaObject; * stateMachine--1..*RootFo
    lder 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 o
    rganizedBy;
8.  }
9.  class ArchitectureView {
```

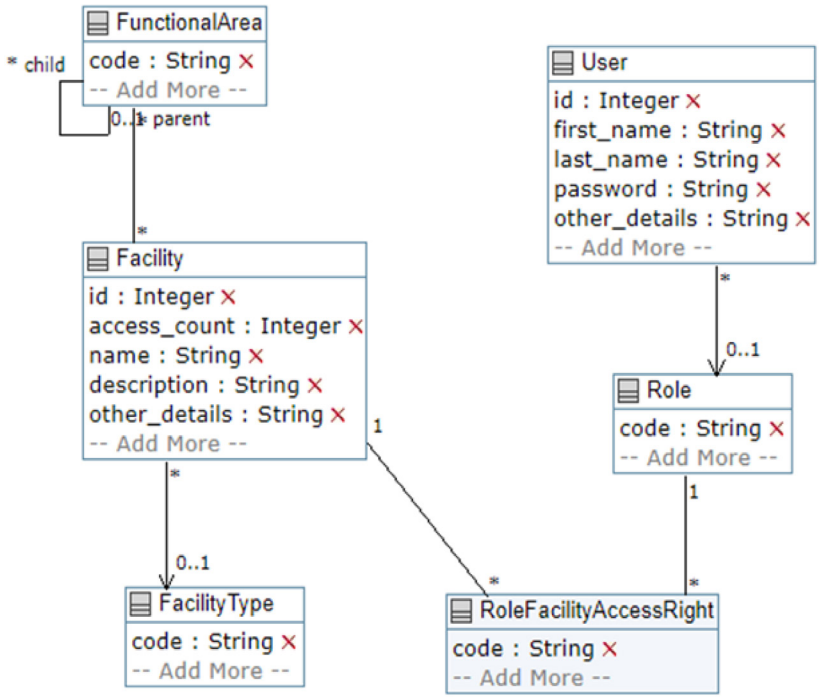

Figure 3: UML Logic diagram access control for medical nanobot.

```
10.     0..1 - > 1..*ModelElement constituents;
11.     1..*views--1..*ArchitectureViewPoint
    conformsTo;
12. }
13. class ArchitectureViewPoint {
14.     0..1 - > 1..*MainView defines;
15. }
16. class MainView {
17.     0..1 - > 1..*ArchitectureView isProjected
    nto;
18.     0..1 - > 1..*ModelElement consistsOf;
19. }
20. class ModelElement {}
21. namespace PrimitiveTypes;
22. class Boolean {}
23. class Integer {}
24. class String {}
```

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 Umple 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.     }
11. } //Functional_Area
12. class FunctionalArea {
13.     String code;
14.     0..1 parent-- * FunctionalArea child;
15.     description {
16.         Operations, Control
17.     }
18.     key {
19.         code
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.     }
```

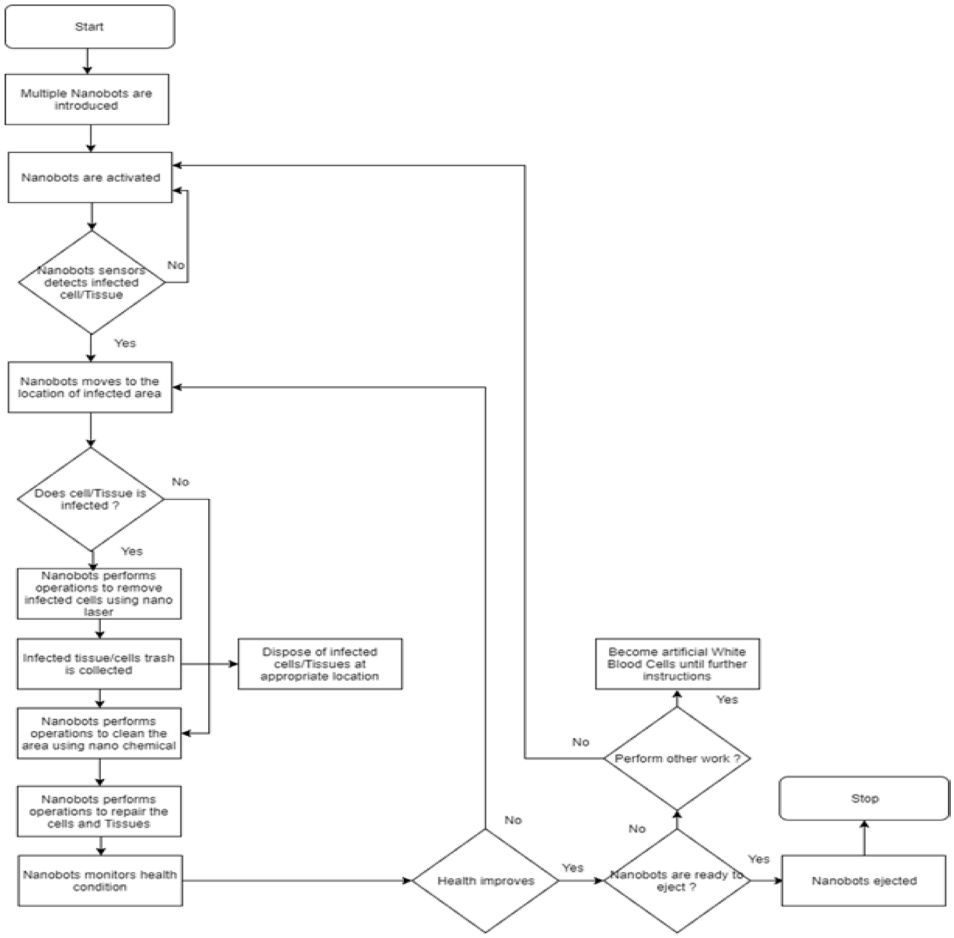

Figure 4: .Flow Chart of a Medical Nanobot for performing various operations inside the body

```
42. }
43. class User {
44.     Integer id; * - > 0..1 Role;
45.     first_name;
46.     last_name;
47.     password;
48.     other_details;
49.     key {
50.         id
51.     }
52. }
53. associationClass RoleFacilityAccessRight
    { * Facility; * Role;
54.     CRUD_Value {
55.         R, RW
56.     }
57. }
```

Author notes

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

1. Solomon N. System and methods for collective nanorobotics for medical applications. 2008
2. Mitra M. Medical Nanobot for Cell and Tissue Repair. Int Rob Auto J. 2017; 2(6): 38.
3. Microscopemaster. Nanobots - Uses in Medicine and Industry - Engineering and Drawbacks; 2018.
4. Jonathan S. How Nanorobots Will Work. Stuff Works; 2007.
5. Bhat AS. Nanobots: The future of medicine. JEMS.2014;5(1):44-49.
6. Moorthy K. Nanobots can check and repair dna. 2016.
7. Miller B, Streeter K, Finn B, Morrison J. C/C++ Programming Guide for the FIRST Robotics Competition. 2008
8. Benson C. How to Make a Robot - Lesson 10: Programming Your Robot. Robot Shop Blog. 2012.
9. Seiko E. Gyro sensors - How they work and what's ahead. 2018
10. Umple Online: Generate Java, C++, PHP, Alloy, NuSMV or Ruby code from Umple. University of Ottawa; 2018.
11. Overview of Robotic Vision - Object Tracking and Image Processing Software. Into robotics; 2013.
12. Solenoid - Let's Make Robots! - RobotShop. Robot Shop.2008.
13. Jacob T, Hemavathy K, Jacob J, Hingorani A, Marks N, et al. A nanotechnology-based delivery system: Nanobots. Novel vehicles for molecular medicine. J Cardiovasc Surg. 2011;52(2): 159-167.
14. Haberzettl CA. Nanomedicine: destination or journey?. Nanotechnology. 2002; 13*(4): R9-R13.*
15. Loscrí V, Natalizio E, Mannara V, Aloi G. A Novel Communication Technique for Nanobots Based on Acoustic Signals. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. 2014; 91-104.