Ứng dụng MFC (Visual C++) trong mô phỏng Robot và hệ Cơ điện tử

Bài 7:
Mô phỏng Robot với đồ họa OpenGL

PHẠM MINH QUÂN

mquan.ph@gmail.com

Nội dung

1. Xây dựng các lớp cơ bản để thiết lập OpenGL và quản lý Robot, Quỹ đạo.
2. Vẽ robot với OpenGL
3. Vẽ quỹ đạo với OpenGL
4. Mô phỏng chuyển động của Robot

1. Xây dựng các lớp cơ bản

- Xây dựng lớp COpenGLControl để hiển thị đồ họa OpenGL trên dialog MFC như trong bài 5,6
- Xây dựng các lớp Robot và Trajectory như trong bài 4

* Có thể thêm các lớp đã xây dựng từ trước vào project mới bằng cách:
  - Copy file .h và .cpp vào thư mục của project mới.
  - Dùng menu “Project -> Add Existing Items…” để thêm các file sẵn có vào project.

* Để trống các hàm vẽ của các lớp Robot và Trajectory

```
void Robot::DrawRobot()
{

}
```

```
void Trajectory::DrawTrajectory()
{

}
```

2. Vẽ Robot với OpenGL

➢ Khai báo và sử dụng đối tượng Robot trong lớp COpenGLControl

File OpenGLControl.h

```
#include "Robot.h"

    Robot Rb1;
```

File OpenGLControl.cpp

```
void COpenGLControl::oglDrawScene(void)
{
    // Clear color and depth buffer bits
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    Rb1.DrawRobot();

    // Swap buffers
    SwapBuffers(hdc);
}
```

2. Vẽ Robot với OpenGL

- Viết code cho hàm DrawRobot()

// File Robot.cpp

```
void Robot::DrawRobot()
{
    GLfloat base_color[] = { 1.0, 1.0, 0.0 };
    GLfloat link1_color[] = { 0.0, 1.0, 0.0 };
    GLfloat link2_color[] = { 1.0, 0.0, 0.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0 };

    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, base_color);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 30);

    GLfloat th=0.02;        // Thickness of link
    GLfloat wd=0.04;        // Width of link
    glPushMatrix();
        glPushMatrix();
            glScalef (0.06, 0.06, th);
            glutSolidCube (1.0);
        glPopMatrix();

        glRotatef(q1*180/3.14, 0.0, 0.0, 1.0);
        glTranslatef(l1/2, 0.0, th);

        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, link1_color);
        glPushMatrix();
            glScalef (l1, wd, th);
            glutSolidCube (1.0);
        glPopMatrix();

        glTranslatef(l1/2, 0.0, 0.0);
        glRotatef(q2*180/3.14, 0.0, 0.0, 1.0);
        glTranslatef(l2/2, 0.0, -th);

        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, link2_color);
        glPushMatrix();
            glScalef (l2, wd, th);
            glutSolidCube (1.0);
        glPopMatrix();
    glPopMatrix();
}
```

// Vẽ base

// Dịch chuyển hệ tọa độ để vẽ link 1
(Do tọa độ gốc khi vẽ khối cube nằm ở chính tâm khối nên phải dịch chuyển hệ tọa độ về chính giữa khâu 1 rồi mới gọi lệnh vẽ cube)

// Vẽ link 1

// Dịch chuyển hệ tọa độ để vẽ link 2

// Vẽ link 2

2. Vẽ Robot với OpenGL

- Điều chỉnh lại điểm nhìn và góc nhìn ban đầu cho phù hợp // File OpenGLControl.cpp

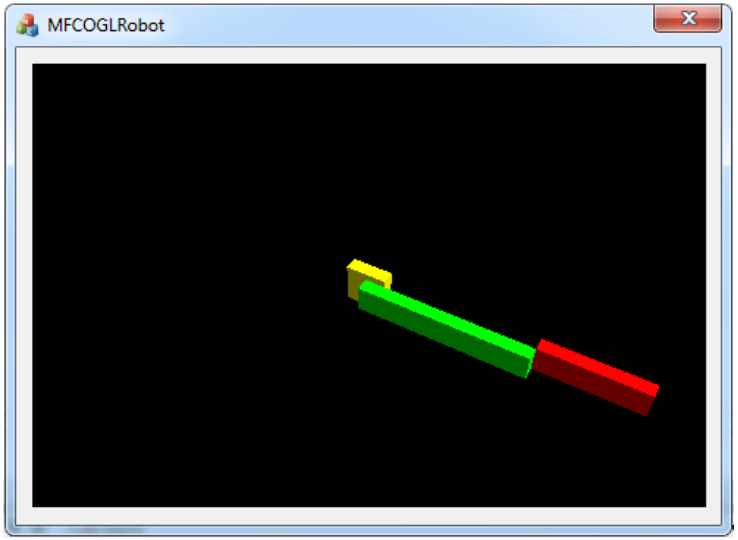
```
COpenGLControl::COpenGLControl(void)
{
    m_fPosX = 0.0f;     // X position of model in camera view
    m_fPosY = 0.0f;     // Y position of model in camera view
    m_fZoom = 1.0f;    // Zoom on model in camera view
    m_fRotX = 45.0f;     // Rotation on model in camera view
    m_fRotY = -30.0f;     // Rotation on model in camera view
}
```

- Thêm câu lệnh sau vào hàm COpenGLControl::oglInitialize() // File OpenGLControl.cpp

```
glEnable(GL_NORMALIZE);
```

// Lệnh glScalef() làm biến đổi vector normal khiến hiệu ứng ánh sáng đối với vật thể bị thay đổi, để tránh hiện tượng này cần kích hoạt tính năng GL_NORMALIZE

Kết quả chạy:



3. Vẽ quỹ đạo với OpenGL

- Khai báo và sử dụng đối tượng Trajectory trong lớp COpenGLControl

File OpenGLControl.h

```
#include "Trajectory.h"

    Trajectory Trj1;
```

File OpenGLControl.cpp

```
void COpenGLControl::oglDrawScene(void)
{
    // Clear color and depth buffer bits
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    Trj1.DrawTrajectory();
    Rb1.DrawRobot();

    // Swap buffers
    SwapBuffers(hdc);
}
```

3. Vẽ quỹ đạo với OpenGL

➢ Viết code cho hàm DrawTrajectory() // File Trajectory.cpp

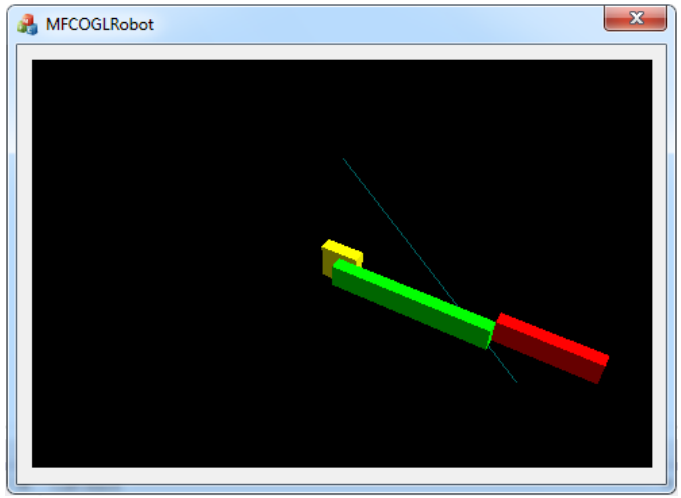
```
void Trajectory::DrawTrajectory()
{
    GLfloat Color[] = { 0.0, 1.0, 1.0, 1.0};
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, Color);

    glPushMatrix();
        glBegin(GL_LINE_STRIP);
        int n =30;              // Number of segments
        for (int i=0; i<=n; i++)
        {
            double u=i*1.0/n;
            glVertex3f(Getx(u), Gety(u), 0.0);
        }
        glEnd();
    glPopMatrix();
}
```

* Thêm các hàm sau đây vào lớp Trajectory

```
double Trajectory::Getx(double u)
{
    return (type==1) ? x0+u*(x1-x0) : cex+ra*cos(u*2*3.14159);
}
double Trajectory::Gety(double u)
{
    return (type==1) ? y0+u*(y1-y0) : cey+ra*sin(u*2*3.14159);
}
```

Kết quả chạy:



4. Mô phỏng chuyển động của Robot

- Thêm hàm sự kiện OnTimer() vào lớp …Dlg

// File …Dlg.h

```
    afx_msg void OnTimer(UINT_PTR nIDEvent);
```

// File …Dlg.cpp

```
BEGIN_MESSAGE_MAP(CMFCOGLRobotDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_TIMER()
END_MESSAGE_MAP()
```

- Thêm biến thời gian t, khởi tạo trạng thái ban đầu khi t=0

// File …Dlg.h

```
    float t;
```

// File …Dlg.cpp, hàm OnInitDialog()

```
    t = 0;
    m_oglWindow.Trj1.SetTime(t);
    m_oglWindow.Rb1.Setpos(m_oglWindow.Trj1.Getx(), m_oglWindow.Trj1.Gety());
```

4. Mô phỏng chuyển động của Robot

➢ Điền nội dung cho hàm OnTimer()

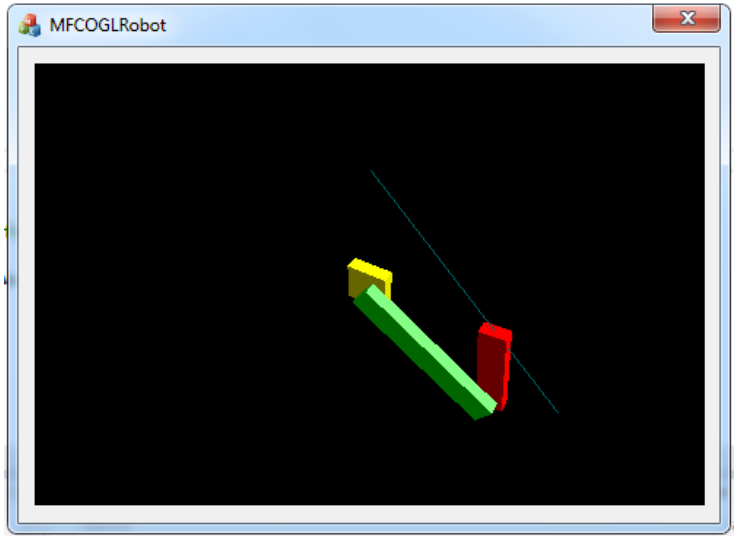
```
void CMFCOGLRobotDlg::OnTimer(UINT_PTR nIDEvent)      // File …Dlg.cpp
{
    // TODO: Add your message handler code here and/or call default
    m_oglWindow.Trj1.SetTime(t);
    m_oglWindow.Rb1.Setpos(m_oglWindow.Trj1.Getx(), m_oglWindow.Trj1.Gety());
    m_oglWindow.oglDrawScene();
    t += 0.1;

    CDialogEx::OnTimer(nIDEvent);
}
```

➢ Thiết lập Timer

```
SetTimer(1,100,NULL); // File …Dlg.cpp, hàm OnInitDialog()
```

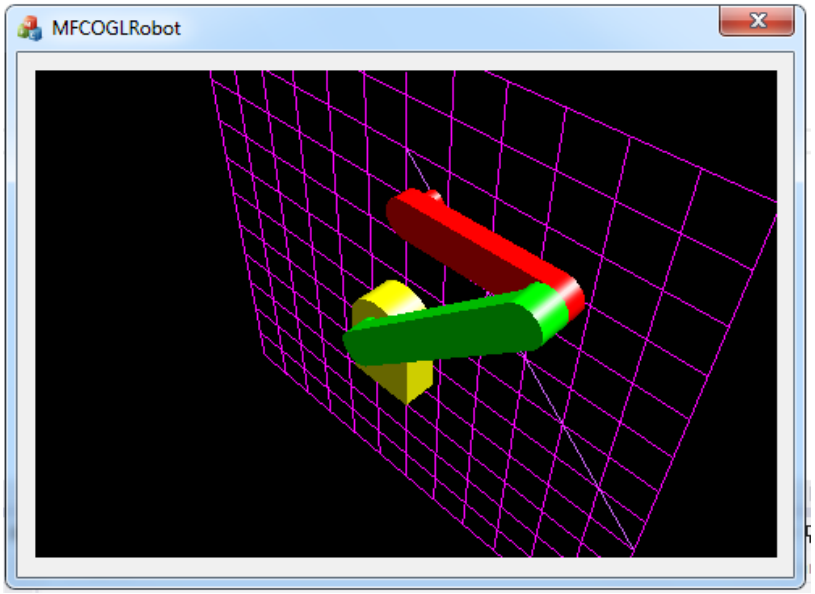
Kết quả chạy:
Điểm tác động cuối của Robot di chuyển theo quỹ đạo

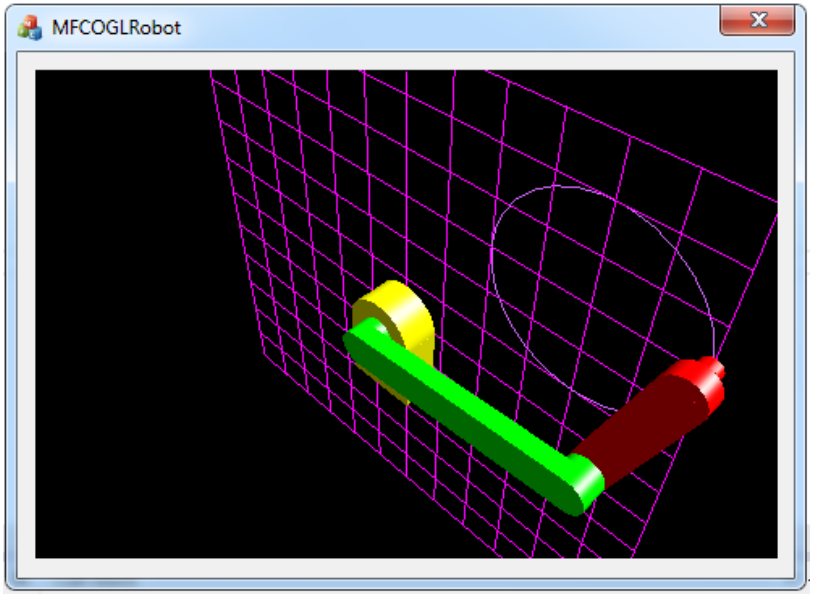


4. Mô phỏng chuyển động của Robot

- Chỉnh sửa hàm vẽ robot, dùng các lệnh vẽ cơ bản của OpenGL để hoàn thiện mô hinh robot

Kết quả chạy:

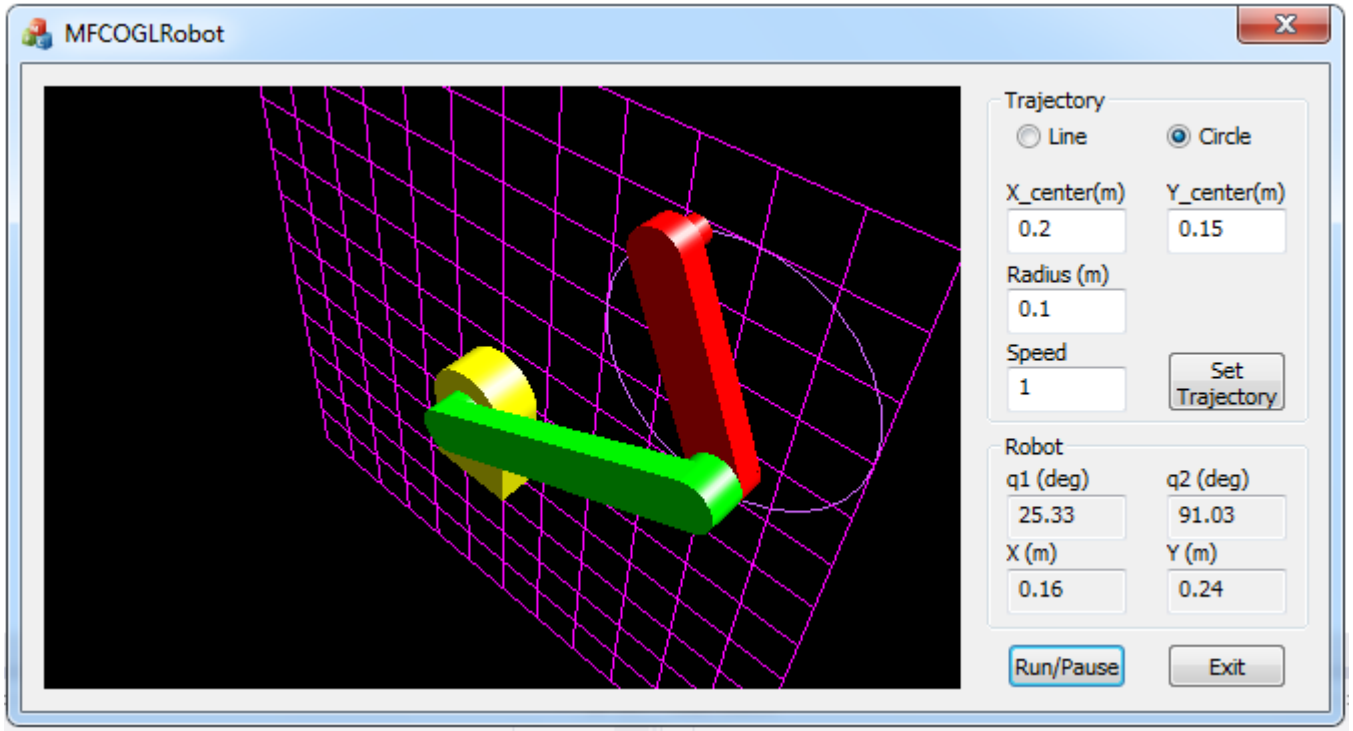




4. Mô phỏng chuyển động của Robot

➢ Thêm các control của MFC để hoàn thiện chương trình

Kết quả chạy:



Hết Bài 7