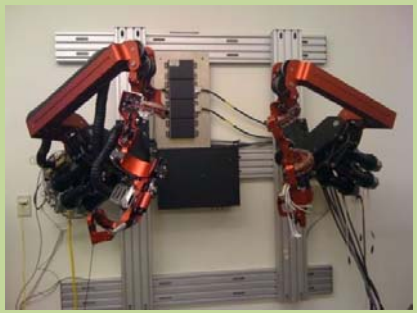
Developing a Virtual Environment For Integration with Exoskeleton Arms

by: Ben Farley Advisor: Jacob Rosen SURF-IT 2009

Baskin Engineering

Background

- The project revolves around a pair of exoskeleton arms
- The arms will be used in rehab for stroke patients
- The goal is to make rehab more interesting and appealing by creating games for patients to play
- To do so, I constructed a virtual environment using Coin3D in which movements of the actual exoskeleton arms move the arms of a simulated entity and allow the user to interact with the environment



The exoskeleton arms used in the project.

Why Coin3D?

Previous work on the project had been done in Microsoft Robotics Studio, but other researchers had used Coin3D for similar tasks with great success, so we decided to try the same.

Coin3D

Pros:

- Good documentation
- Easy, flexible graphics

Cons:

- No built-in physics

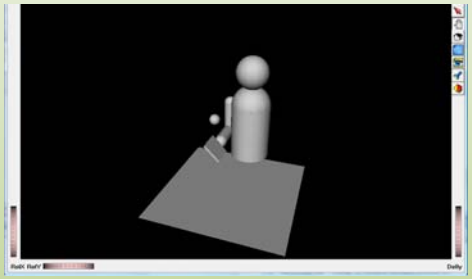
Microsoft Robotics Studio

Pros:

- Built-in physics

Cons:

- Poor documentation
- Focuses on gaming – physics sacrifice accuracy for speed



The entity

- Basic geometry objects to represent torso and head
- Complex arm – spheres to represent the shoulder, elbow, and wrist joints, and cylinders to represent upper and lower arm. Hand shape can be changed based on the intended purpose of the simulation
- Rotations can be applied to each joint of the arm to allow it to be placed in the same position as the arms of the exoskeleton

Moving the arms

To make the virtual entity's arm mirror movements of the exoskeleton arms, the program performs the following steps:

- Connect to the arms using sockets
- Periodically receive an array of bytes from the arms
- Interpret the bytes as floats, which correspond to the seven joint angles of the arm
- Update the angles of our simulated entity based on the angles we've received

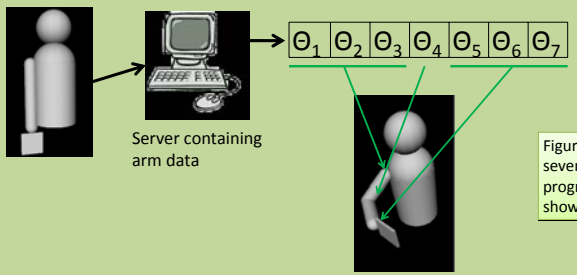


Figure 1. The arms send seven angles to the program, in the order shown.

Physics engine

- Simple world – ball, list of objects it can collide with
- Objects are infinite planes, with bounding planes to test collisions
- Find distance between ball and all objects, if $D < 0$, they are colliding
- If colliding: $F_{net} = F_{grav} + F_{normal} + F_{fric}$
- If not colliding: $F_{net} = F_{grav}$
- Use Newton's Second Law to find velocity and position based on net forces
- Focus on accuracy of calculations rather than speed

The simulation

- The simulation must repeatedly do two things: connect to the robot and redraw the scene to reflect the movement of the arms and ball
- These need to run at different speeds
- Solution: C++ threads

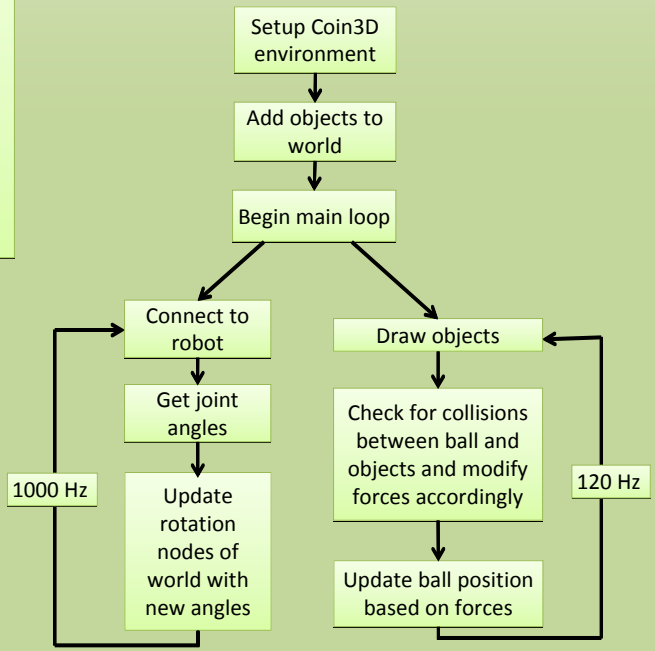


Figure 2. The flow of the program.

Future work

- Improve accuracy of physics engine by modifying our collision-detection algorithms
- Add the ability to move a second arm
- Implement a collection of games that rehab patients can play