Robot Programming
Introduction

The interface between human user and an industrial robot is extremely important.

Manipulators are a minor part of an automated process. So manipulator programming is considered within a workcell.
An Assembly Task Example:

*Put a washer on the bolt* requires the performance of subtasks such as:

1. Move the end effector to the washer supply.
2. Grasp a washer.
3. Move the end effector to the end of the bolt.
4. Orient the washer so it is perpendicular to the bolt.
5. Translate the washer so the axis of the bolt passes through the center of the hole in the washer.
6. Translate the washer along the axis of the bolt.
7. Release the washer.
8. Retract the end effector.
Levels of Robot Programming

- Teaching by showing
- Explicit programming languages
- Task level programming languages
Teaching By Showing

One of the simplest programming methods involves leading the robot through the desired motions, and recording the joint motions for later playback.

The robot motions can be input by physically moving the robot end effector manually, or by moving the robot using a hand-held teach pendant.
Teaching By Showing

- The robot joint positions can be either recorded automatically at some sampling rate (e.g., for continuous motions such as arc welding), or recorded manually by the operator (e.g., for pick-and-place tasks).
Teaching By Showing

- Once the motions have been saved, the controller simply plays them back by using the recorded joint values as the control inputs.

- This method is relatively simple. However, this approach is not well suited to complex tasks, and the programs are difficult to modify without starting over.
Teaching By Showing

Limitations:

- The controller must have sufficient memory to store information on the data points (usually both joint and Cartesian space).

- Secondly, in the case that the robot motor is inoperative, the operator must overcome the weight of the motor as well as the friction that exits in the arm joints and gears.
Explicit Programming Languages

Robot programming languages (RPL) have special features for manipulator programming.

3 categories:
- Specialized manipulation languages.
- Robot library for an existing computer language.
- Robot library for a new general purpose language.
Specialized Manipulation Languages

- These robot programming languages have been built by developing a completely new language which addresses robot specific areas.
- An example: VAL II, developed by Unimation inc.
In addition to being a sophisticated robot programming language, VAL II is a complete robot control system. It is designed to readily communicate with other computer-based systems such as vision and tactile sensors. One can learn to program the robot simply by looking at some example programs and studying the instructions given in the editor.
VAL II

Typical commands:

# Where: display current robot position (joint and world space)
# Here: define a position
# Move: move to a specified location
# Delay: pause for a number of seconds
# Calibrate: calibrate the robot
# Execute: run a program

Typical program:

A  Move PT1
Delay (10)
Move PT2
Goto A
Robot Library for a New General Purpose Language

These robot programming languages have been developed first by creating a new general language as a programming base, and then supplying a library of predefined robot specific subroutines.

Example: AML, KAREL.
Task Level Programming

- Allow the user to command desired subgoals of the task directly.
- Have the ability to perform many planning tasks automatically.
- Example: “grasp the bolt.”
- Does not exist yet!
Requirements of a Robot Programming Language

- World modeling
- Motion specification
- Flow of execution
- Programming environment
- Sensor integration
World Modeling

The most common element of RPL is the existence of geometric types:

Geometric Types

- Joint angles
- Cartesian Positions
- Orientation
- Frames
World Modeling

All motions are described as tool frame relative to station frame. Goal frames are constructed from expressions involving geometric types.

Names for various geometric types

Shape of objects are not included
World Modeling

Affixment:
Some language allow two or more named objects to be affixed. If one is moved the others are moved too.

CAD data.
Some world modeling systems allow CAD models to be included.
Motion Specification

- A basic function of any RPL is to allow description of robot motion by motion statements.
- These statements specify: via points, goal points, interpolated motion or Cartesian straight line motion, space and time control.

Move goal1
Move via1
Motion Specification

Other features.
- The ability to do math on frames, vectors, matrices.
- Different geometric representation and conversion between them.
- Constraints on speed.
- The ability to specify goals relative to various frames.
Flow of Execution

Most RPL support concepts such as:

- Testing, branching, looping, call to subroutines and interrupts.
- Parallel processing for control of two robots or other equipments in the same workcell in a parallel fashion. (By using signal and wait primitives).
- Event monitoring by interrupts or though polling.
Manipulator programming is difficult and tend to be very interactive, with a lot of trial and error.

The productivity of “edit-compile-run” would be low hence most RPL are *interpreted*: statements run one at a time during program development and debugging.

Typical program support: text editors, debuggers, and file system.
Sensor Integration

Any RPL should have capability to query touch and force sensors. Some may support other sensors:

- **Vision**: to extract coordinate of objects
- **Conveyor belt interface**: to track belt motion and acquire objects from the belt as it moves
- **Active force control**: the ability to specify force strategies, display force data and programming the stiffness

*Move arm to goal*

*With force=20*ounces along z*
Problems Peculiar to Robot Programming Languages

- Internal world model versus external reality
- Context sensitivity
- Error recovery
Internal World Model Versus External Reality

Discrepancies between internal model and external reality will result in poor or failed grasping of objects and collisions. This correspondence must be established in the program’s initial state and maintained through its execution.

For dealing with position uncertainty and manipulator accuracy it's essential to be able to refine the position information.
Internal World Model Versus External Reality

- Some times sensor information (touch, vision and force) is used for dealing with uncertainties.
- Debugging is difficult because retrying of more than one action might be needed.
- Physical change to objects (a painted surface, a drilled object) are not reversible and the object must be changed.
Context Sensitivity

In RPL *Bottom-up-programming* doesn't always work well.

Manipulator programs are highly sensitive to context, motion speed and initial condition.

Programs tested in low speed may experience greater servo error or need force strategy change in higher speed.
Error Recovery

- In assembly operations the objects might not be exactly where they should be and hence motion may fail.

- Error detection is difficult because robot has limited sensing and recovery capabilities. The program must have explicit test.

- Position test:
  - For jamming detection (if no change).
  - Missing object or slipping if too much change.
Error Recovery

- Checking for every motion is cumbersome so only most likely to fail motions are being checked.
- Code to recover fail is a major part of manipulator program.
Typical Program Development Steps

- Workspace is set up and its components (robot, feeders, conveyors, jigs etc.) are fixed.
- The location (orientation and position) of the workspace components are defined using the data structure provided by the programming language.
- The task is partitioned into a sequence of actions which can be implemented by the programming language (e.g. Move, pause, repeat etc).
- The program is refined and debugged.
Robot workcell Simulations and Off-line Programming (OLP):

- Interactive, 3D graphic simulation packages for designing, evaluating, and off-line programming robotic workcells are available: IGRIP, GRASP, workspace.

- Actual robotic/device geometry, motion attributes, kinematics, clamps, fixtures, and I/O logic are incorporated to produce extremely accurate simulations.
Robot workcell Simulations and Off-line Programming (OLP):

- Interactive 3D graphics to verify production concepts, workcell designs and manufacturing processes before implementing them on the shop floor.
- After verification is completed, automated factory floor devices, such as robots and turntables, can be programmed off-line based on the CAD data for the part being processed.
Robot workcell simulations and Off-Line Programming (OLP):

- Includes built-in libraries of robots, common production equipment, and application-specific options that include tooling such as spot weld guns, and even paint spraying tools.

- These libraries, an integrated CAD system, and data translators for IGES, DXF, pro/engineer and other file formats support the rapid development of accurate simulation models.

- Simulation and analysis functions include automatic collision and near-miss detection, and automatic adjustment of a robot work envelope for tool offsets and joint limits.
Robot workcell Simulations and Off-line Programming (OLP):

screenshot of IGRIP, by Deneb.
Next Course:

Student Presentations

*Introduction to Robocup*

*Amirkabir University of Technology*
*Computer Engineering & Information Technology Department*