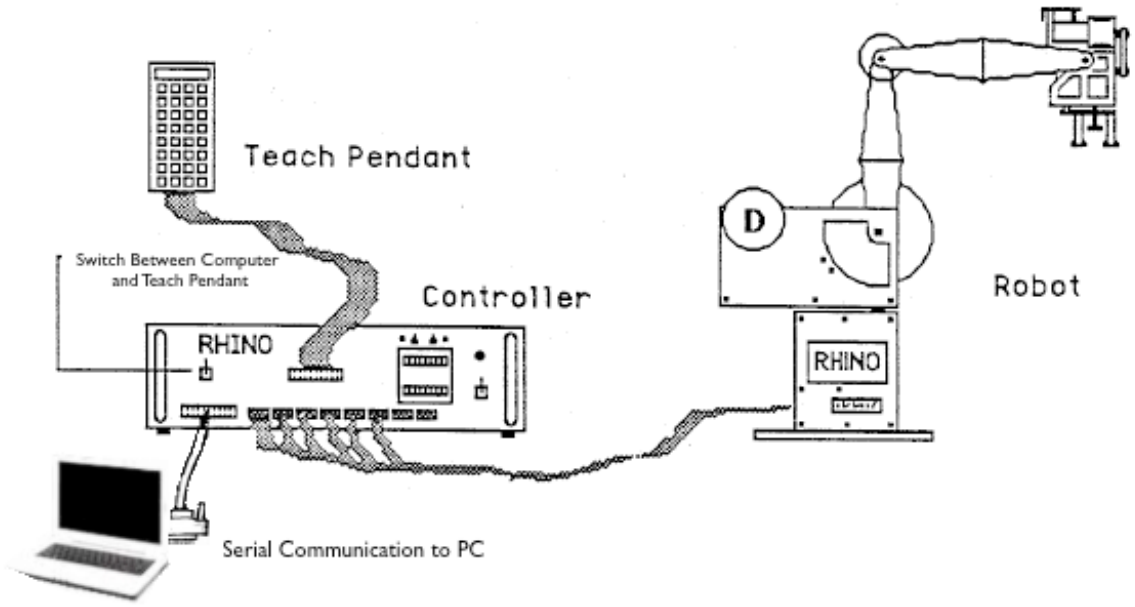
Rhino Arm

Jeff Caley
12-8-2010
ECE - 578

Introduction:
The RhinoArm project consists of a XR-3 Rhino Robotic Arm made by Rhino Robotics, a Mark III Controller and a robotic arm simulator build in Unity using A* search to solve the inverse kinematic problem. The user is able to give the software program a XYZ coordinate and have a simulated model of the XR-3 arm move to that position in space. The Physical XR-3 arm will also move with it.

The System:
To make the RhinoArm operate; an XR-3, a Mark III Controller and a Computer are needed. The XR-3 connects to the Mark III through 10 pin connector ribbon cable. The cable connects directly to each servo and controls the movement of the robotic arm. The controller gets its commands from a computer via the serial port. The serial port is RS-232 and is of the 25 pin variety. The Teach Pendant is an alternative way to move the robotic arm, with different buttons moving each servo independently.



The System

XR-3:
The XR-3 is a five axis robotic arm. Each axis moves independently and each can be controlled simultaneously. There are six servo motors, 5 to control each axis and a 6th to open and close the gripper, with incremental optical feedback to allow precise movement. Each drive servo is connected to the controller by a 10 conductor ribbon cable which supplies power and carries position data from the encoder. This position data is fed into the Mark III controller and is used to position each arm to the requested location. The robot can lift a 1kg object and has a reach of 24 inches from its base. Each of the six motors is associated with a specific letter of the alphabet for naming; A-

F. The ranges of motion for each motor are as follows:

Motor "F" Body Rotation - 350 degrees
Motor "E" Shoulder Rotation - 210 degrees
Motor "D" Elbow Rotation - 265 degrees
Motor "C" Wrist Rotation - 310 degrees
Motor "B" Gripper Rotation - +/- 7 revolutions
Motor “A” Gripping Action - N/A

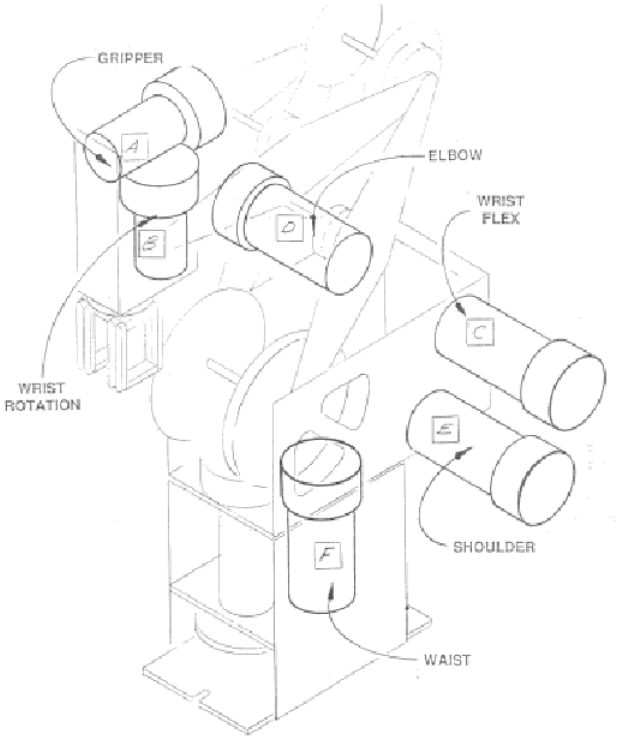


Figure 1: Motor Placements

Figure 1 shows the location of each motor and what joint they operate. Each motor also has a corresponding microswitch which is used in setting the home position. At a certain point along the movement of each axis, the microswitch will be closed. The Mark III controller can register when these switches are closed and this information can be used to determine when the robot is in a specific state.

The Optical encoders.
The optical encoders are disks attached to each motor with small holes punched in them. As the motor spins, the disk also spins and an optical device is able to count the number of times it sees a hole in the disk. This is then fed into the microcontroller and is used to keep track of how much each motor is being moved. However, the optics are not able to determine the starting point of each motor, so it is only able to say how much it has moved, not where it has moved to.

Mark III Controller:
The Mark III Controller is the brains of the XR-3 robot. It comes with a built in microprocessor, operating system and electronics for control. This controller allows for a direct interface to the 8 servo motor outputs, the attachedTeach Pendant (not used in this project) and the use of the standard RS-232 serial port communication.

Talking via Serial Communication:
The Mark III Controller uses a 25 pin RS-232 standard serial port to make communication to a computer. It doesn’t use a handshake and requires a direct pin connection setup as seen in Figure 3. Most computers these days don’t have a 9 pin serial port and almost all won’t have a 25 pin serial port, so a USB to serial adapter can be used. This project uses a TRENDnet TU-S9 USB to Serial adapter with drives available at http://www.trendnet.com/asp/download_manager/list_subcategory.asp?SUBTYPE_ID=571. A 9-25 pin adapter was used to increase the pin number to 25 and

then a 25-25 pin null modem cable is used to get the appropriate pin connections needed to communicate with the Mark III. The settings for the serial communication are a 9600 baud rate, 7 data bits, 2 stop bits and even parity. It uses ASCII code and has been programmed to operate 14 basic commands. The commands are as followed:

<return> : Carriage return (initiate a move)
? : return distance remaining
A-H : set motor movement value
I : Inquiry command (read limit switched C-H)
J : Inquiry command (read limit switched A-B and inputs)
K : Inquiry command (read inputs)
L : Turn Aux #1 port ON
M : Turn Aux #1 port OFF
N : Turn Aux #2 port ON
O : Turn Aux #2 port OFF
P : Set output bits high
Q : Controller reset
R : Set output bits low
X : Stop motor command

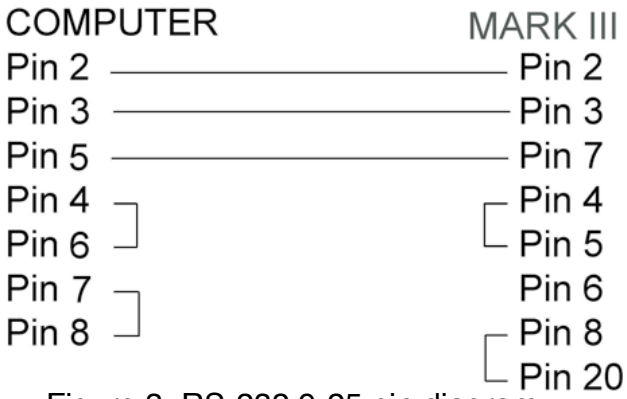


Figure 3: RS-232 9-25 pin diagram

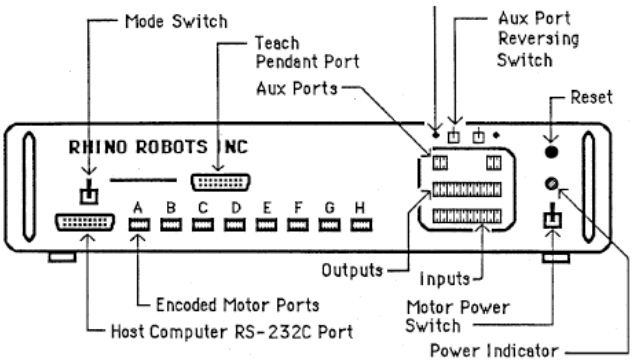


Figure 2: Mark III Controller

Here is an example of how this works. If you want to have the E motor move 80 optical encoder units of movement, simply send the ASCII code “E80” through the serial port. This translates to about 8 degrees of movement. If you sent “E?” while the robot is moving, it reports how much more it has to move before it reaches its destination. The controller accomplishes this by taking the inputted number and adding it to a “error” register. The controller is constantly trying to keep the error register at 0, so if 60 is added to the error register for motor C, Motor C will start to move to shrink the error in this register. After 60 optical encoder movements are recorded, the error register will be equal to 0 again and the motor will stop moving.

Inverse Kinematics Problem: The inverse kinematics problem states "Given the desired position of the robot's end effector, what must be the angles at all of the robot's joints?" To put it another way, given an XYZ coordinate, how much does each joint have to rotate to move the tip of the robotic end effector to the given location. Not only that, but what route is the shortest to the goal location? In this case, shortest means the least number of degree changes for each joint. Often with robotic arms that have multiple axis of rotation, there is more then one solution to the inverse kinematics problem; meaning there is more then one set of angles that gets the robotic arm to a desired position. But

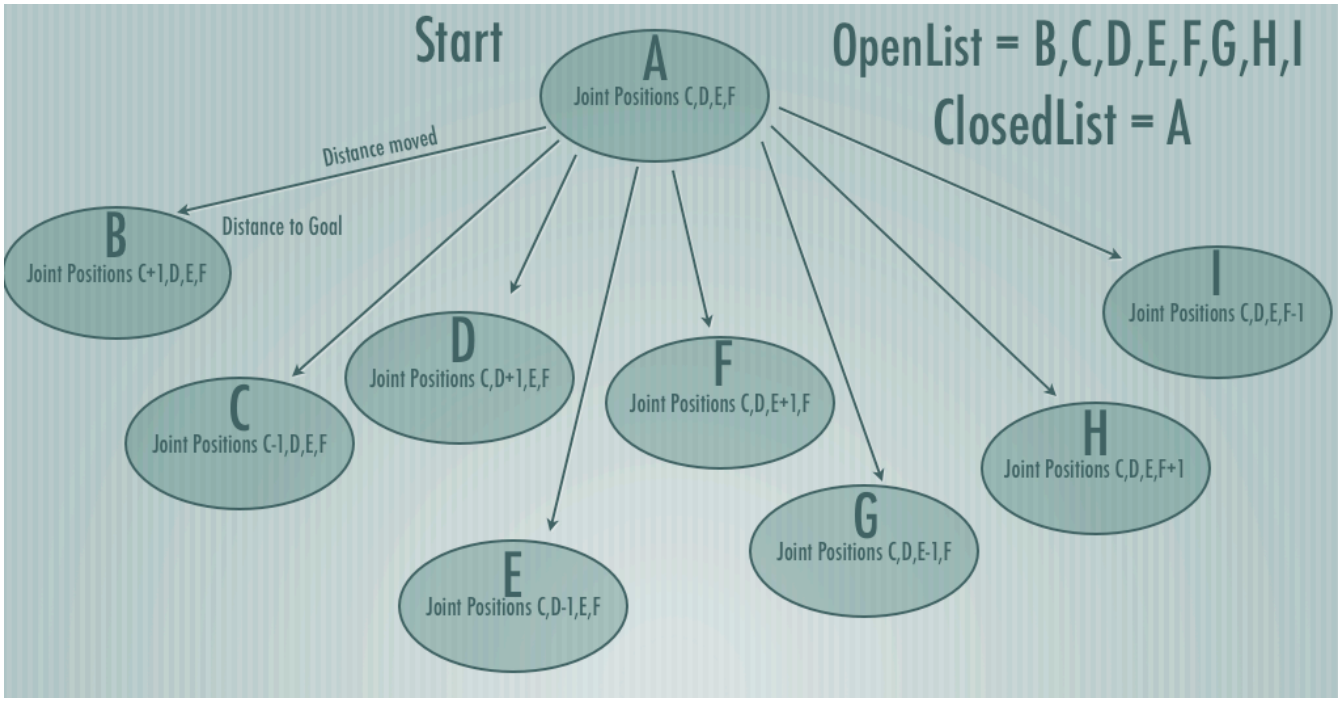
looking for the most efficient solution can be important and challenging. This all boils down to a large search problem.

A* Search:
A* Search is a algorithm that uses a best-first search and finds the least-cost path to the goal. It does this by using past cost to get to current position plus an estimated cost to the goal. g(x) denotes all the previous costs to get to the current position. h(x) is the estimated future cost to move to the goal. This heuristic has to be invented by the user of A* and requires some creativity and intuition about the problem. f(x) is g(x) + h(x) and is minimized to find the solution.

A* Search implementation to solve the inverse kinematics problem:
To solve the inverse kinematics problem, A* search was used. The initial position of the 4 joints; C,D,E,F, are stored in a node on an open list. The first element of the open list is then expanded to show all next state positions of the robotic arm. Because there are 4 joints, there are 8 possible next states in which the robotic arm could exist. Each of the 8 children states have only one joint incremented or decremented by one degree. These new 8 states are all added to the open list and the closed list. The closed list is used to make sure that no state is ever entered twice. Each time a node is opened and the next possible states are evaluated, they are compared to all the states on the closed list. If any of the states are already on the closed list, then they are not added to the open list. The cost to move the robotic arm to each of these new states must also be

factored in. The first implementation I tried defined the cost as the distance from one state to the next in Euclidian distance. As we traverse the tree, these costs continue to add to themselves to give us g(x), the total cost to the current node. h(x); estimated distance to goal, was defined as the Euclidian distance from the current state to the goal position. With g(x) and h(x) now defined, we can find the sum of the two for each node and sort the open list. The node with the smallest sum is currently our best path to our goal. This node is then taken and expanded into its next 8 possible states. This is repeated over and over again until the distance to the goal is equal to zero. At this

g(x) = Total Distance Moved
h(x) = Euclidian Distance to Goal
f(x) = Total Estimated Cost
f(x) = g(x) + h(x)



point, the goal will be reached. By walking back up the chain of the nodes that have been expanded, you can see the route the A* search has taken. The final node will have a specific value for joints C,D,E and F that will represent the arm's finally position that will be touching the goal.

Results for initial Implementation:

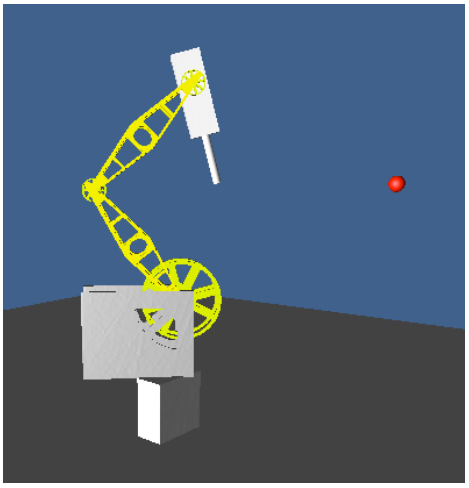
After implementing this search technique, a flaw was found in the heuristic. Because the estimated cost to goal is a perfectly straight line, and the robotic arm moves in arc's, the robot can't traverse the exact path that is estimated. Because of this,as the current state traverses down the tree, f(x) increases because total distance moved is not in a straight line, but estimated distance to goal is. This causes nodes near the top of the A* tree to always have a smaller f(x) then nodes just a few levels lower. This keeps the search space large and doesn't do a good job of finding the solution. However, if the search ran long enough it would find the solution. The second issue with this implementation is what this search is minimizing. Our goal was to minimize the number of angle changes for each joint, but this search is trying to move the end effector along the straight line path to the goal and minimize the distance off this straight path. Because of these issues, a second heuristic was tried.

2nd Heuristic tried:

To compute how to minimize the number of degree changes needed to reach the goal, a new heuristic was tried. This new implementation used a kind of A* hybrid. Instead of using a traditional $f(x) = g(x) + h(x)$ formula, f(x) is equal to the average Euclidian distance moved per node traversed. The distance moved is only in a straight line towards the goal. This heuristic will find the minimum number of angle changes needed to reach the goal, but it also struggles to find the solution if the first few moves needed to reach its goal do not move towards to goal. Because of this, certain starting states perform much better then others.

Unity Simulation:

Unity is a game development tool that can be used to create fully immersive games. For this application it was used to create a 3-D interface and simulation of the XR-3 robotic arm. The Rhino arm was drawn in CAD software and then imported into Unity for scripting. Unity 3.1 was used for this simulation and is available for download at http://unity3d.com/unity/download/. Unity breaks up the programming into Scene and Camera. The scene shows your 3-D objects and executes the attached scripts, the camera displays these objects from a specified angle with a specific lighting. The robotic arm simulation consists of five CAD drawing that are placed in space in the scene. A robot controller script specifies where to place each of these drawings and at what angle they should appear. A sixth item is in the scene, this being the goal state. It appears as a small red ball. An A* search script then interacts with the scene. Between each frame, the A* search script sorts its open list, grabs the node with the smallest f (x) , adds it to the closed list and then opens the node. Eight new positions are found and added to the open list. The list is sorted and the node with the smallest f(x) sends data to the robot controller script to tell it how to position each of the 5 CAD drawings representing the robotic arm. This results in the simulation showing the movement of the robotic arm for each step as it moves through the A* search algorithm. Another version of the program only drawn a new scene when the goal state has been reached. After the solution has been found, the Unity program exports the object that holds the joint data to an XML file. This file is named finalstate.xml. This file is then imported into a program to move the physical robot arm.



Software to control the XR-3:

To control the XR-3, software was written in MonoDevelop (available for free download here http://monodevelop.com/) using C#. It requires the exported finalstate.xml to be in the same directory as the serialComm software. The program starts by importing the XML file and reading in the final joint locations of the solved robot. An initial setup sets the movement commands for each joint in the correct direction depending on the direction that each joint will need to move off the home position. The home position of the robot is a 0 degree angle on the base, a -20 degree angle of the shoulder off vertical (if the arm was pointing straight up, all joints would have a 0 degree angle), the elbow at 90 degrees and the wrist at 90 degrees. Reading in the joint values allows us to determine how many optical encoder moves will need to be made for each joint. We then loop through 4 different loops each increases the motor error register by one until each motor has moved to the appropriate location. We then pause for 10 seconds until moving towards home. Home is found by moving in the opposite direction for each of the joint moves. Between each optical encoded movement, the “I” register is polled to see if any microswitches have been tripped. Once the appropriate microswitch for that specific joint is tripped, that loop is finished and the next joint starts the same process. After each joint has moved back to home, the program ends.

Future work to be done:

- Add Stereo camera
- image processing to identify items
- Locate items in XYZ plan and have arm pick them up: (this would mean more logic would need to be added to the A* search to allow for the gripper to come in at an appropriate angle to pick items up)
- Avoid objects and recalculate A* search with now seen objects
- Try other Search algorithms and compare to A*

Sources

- http://www.rhinorobotics.com/
- http://ifac-mmm-tc.postech.ac.kr/artmind/library/download.php?db=class_eecs565&no=1&s_id=0 (A great manual on how to work the XR-3 and the Mark III)
- http://unity3d.com/
- The Rhino Robotic Arm uses manual (Found in the lab)
- www.monodevelop.com
- Samuel Grant Dawson Williams, Using the A-Star Path-Finding Algorithm for Solving General and Constrained Inverse Kinematics Problems., 2008. Available from: http://www.oriontransfer.co.nz/research/Inverse%20Kinematics%20A-star.pdf