Scorbot-ER 5Plus Integration with LTLMoP

Jamie Sternlicht
jas734@cornell.edu
Verifiable Robotics
Cornell University

December 18, 2013

Abstract

The purpose of this project was to integrate the Scorbot-ER 5Plus with LTLMoP (Linear Temporal Logic MissiOn Planning). The Scorbot-ER 5Plus is a robotic arm with a two-prong gripper, made by **intelitek**. By operating the Scorbot-ER 5plus with a RS 232 adapter and creating a series of handlers for LTLMoP, the Scorbot was effectively navigating in the XY plane and implementing actuators upon activation of a dummy sensor.

1 Introduction

The purpose of this project was to integrate the Scorbot-ER 5Plus with LTLMoP (Linear Temporal Logic MissiOn Planning). The Scorbot-ER 5Plus is a robotic arm with a two-prong gripper, made by **intelitek**. The Scorbot comes with a controller, and two software control packages as explained in its manual (which is copyrighted 1996). SCORBASE and ATS are the two software packages it comes with. Scorbase is a Graphical User Interface (GUI) which allows you to manually control the joints, save gripper positions, and write functions based off of saved positions. Advanced Terminal Software (ATS) is a terminal like interface, where you can type direct commands to the robot. The commands used in ATS can be used on a personal computer by connecting to the serial port using an RS 232. Ultimately, I was able to create a specification in LTLMoP, to have the Scorbots gripper navigate on a 2D plane. Using a dummy sensor (as the Scorbot itself only has encoders for sensors), the Scorbot could navigate regions, and upon "sensing" a target, drop down and grasp it, and return it to the goal.

2 Initial Work

We collected the Scorbot-ER 5Plus at the end of May 2013. It had been in severe disuse and needed some care to get it working again. ASL luckily received it with a computer (running Windows XP version 2002) with the corresponding programs of Scorbase and ATS, so we could get to testing the Scorbot out right away. Upon turning on the Scorbot controller and its motors, we were able to open up Scorbase and hit the "homing" button. The Scorbot needs to be homed every time it is booted up, otherwise it will not recognize commands.

This first attempt at homing did not work. I was able to narrow down the issues to the microswitch contacts. When the Scorbot "homes" it is activating a series of five (5) microswitches, which if any are not activated, cause homing to fail. Once the Scorbot cannot find one of the microswitches, it fails homing, reports which axis failed and quits the homing function. Initially, the elbow axis was failing, which required a readjustment of its bearing housing cover (see Manual 9-3, item 1 for reference). Once the bearing housing cover was rotated to make contact with the microswitch, it was able to pass

the elbow axis, where I encountered my next issue. The wrist roll was now not homing properly. This was fixed by loosening the screws on the front and back face of the gripper assembly and raising the assembly so it better engages with the wrist gears. Once the homing issues were fixed, the Scorbot was operating relatively normally, albeit a little bit creeky sounding.

The Scorbot could now home and move manually using Scorbase and ATS. The Scorbot can move manually, adjusting each joint or movement in the XYZ Pitch and Roll. It could also save positions to send the gripper to later. It could also save offset positions based on encoder counts for the joints or millimeters for XYZ movements. Saving an offset position meant that the Scorbot asked for encoder counts or millimeters for each axis, and then when that offset position was called later, it would move that respective amount (as long as it was an attainable position).

Upon further investigation, it was discovered that connection to the Scorbot using a USB to Serial Port adapter (RS 232) was possible. Using the "screen" command in terminal we could directly connect to the Scorbot, and send commands that had been used in ATS. This made moving forward hopeful as controlling the Scorbot through a personal computer was possible.

3 Experimentation with ROS: MoveIt!

In the early phases of this project there was the ambition to integrate **ROS: MoveIt!** with the Scorbot and LTLMoP. MoveIt! is an open source *"advanced software for mobile manipulation, manipulation and motion planning"*. The idea of using MoveIt was to have a visual motion planning program, which once a model is uploaded, the program would do most of the kinematics. With that, then we could simply send joint angles to the Scorbot and MoveIt! would work with the motion planning.

While installation seemed to proceed smoothly, upon trying to operate MoveIt! using its tutorial, I ran into issue after issue. Sometime early June 2013 Gazebo became independent of ROS, causes a series of issues when trying to operate Gazebo in conjunction with MoveIt!. Trying to solve my issues with the program independently was not fruitful, and resorted to asking questions on a forum. While waiting for response, I proceeded to attempt to modify

a Scorbot 4 model which was provided by another Scorbot Forum member. See Figure 1 below.

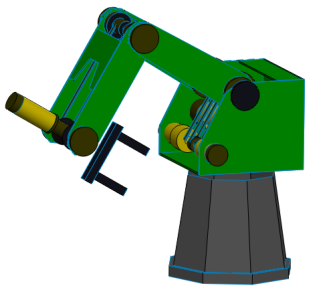

Figure 1: Scorbot CAD imported and edited in SolidWorks

Work with ROS: MoveIt! was discontinued in order to focus on LTLMoP integration with the Scorbot.

4 LTLMoP

In order to proceed with setting up LTLMoP with the Scorbot, it was first necessary to have a basic understanding of Python. Never having coded in Python slowed things down, but after learning the basics, and learning some things about PySerial (which allows you to connect to a serial port), I had the foundation to start creating handlers for LTLMoP. Before creating my own handlers, I experimented with python scripts, controlling the robot to move from pre-defined position to pre-defined position, also to see how well a PySerial could connect to the port to find pose, or move to an offset position.

4.1 Motion Control Handler

The motion control handler for the Scorbot is essentially vector contro. Rather than finding the velocity and angular velocity (since these are functions we cannot control on the Scorbot at this time) I took the velocity vectors of X and Y that vector control created. The vectors of X and Y

that vector control creates are very small increments, so I gave them a large multiplier (200) so that the Scorbot would move at larger increments. These X and Y values are then sent to the Drive handler, so that it can create a command to send the Scorbot to an offset position.

4.2 Drive Handler

Using the X and Y values, the Drive handler has the Scorbot create an offset position. Figure 2 shows what is going on in the serial port. The Scorbot didn't need to move in the Z direction, or change Pitch and Roll, these offset positions remained 0. X and Y as shown would have values inputted into them, using a write to the serial port.

```
> teachr oset
   X [.] > 200
   Y [.] > 200
   Z [.] > 0
   P [.] > 0
   R [.] > 0
```

Figure 2: Teaching the Scorbot an offset

First the Scorbot had to receive the command that it was going to create an offset position. By writing to the serial port the "teachr oset" command, it also had to read back the echo of "teachr oset". For every line that was then created by the serial port, each line then had to be read back, otherwise it could lead to interferences where sections of commands were written into a line being read. So as the X and Y values were sent to the Scorbot, a line had to be read after each value written, including the zeros that were being written to Z, P and R. When incorrect information was being written to the serial port, (which is what would happen if only parts of command were being written at a time because of reading interferences) the Scorbot would function out of luck, and eventually lead into error. Originally this was thought to be an issue with the Drive handler and the Pose handler interfering, by commenting sections of code on both handlers, we were able to see that neither were overlapping. By printing exactly what was being

shown in the readlines, we could see that lines were being interrupted. So therefore, by reading every line that would be printed by the Scorbot before writing a new command, I was able to resolve this issue.

4.3 Locomotion Handler

The Locomotion handler was much simpler than either the Motion Control or Drive handlers. The Drive handler sends the composed offset command to the Locomotion handler, which then writes it to the serial port.

4.4 Pose Handler

Despite the lack of sensors on the Scorbot, it has incredibly accurate encoder count tracking. Meaning if at any point in time we wanted to find pose, it reports an extremely accurate XYZPR position relative to its zero. While this is great, it's not incredibly useful in LTLMoP, and in such a small map, because an action from the Drive handler needs to complete before it can call pose, and pose needs to finish being found before the Drive handler can begin again. This causes a pause in navigation, every time the Motion Control handler needs to calculate a new vector it needs to call its pose, create a new vector, send the new offset to the Drive handler.

```
>listpv pose
   1:0        2:1791     3:2746     4:0        5:-1
   X:1690     Y:0        Z:6011     P:-636     R:-1
```

Figure 3: Finding pose from the Scorbot

As you can see, the Scorbot receives pose in terms of its joint axes encoder counts, as well as millimeters in the XYZPR. As we were working in XY, we purely wanted to receive this information. So the handler had to read in each of the lines that the "listpy" command sent back, but only sort out the X and Y information.

4.5 Actuator Handler

Navigation in LTLMoP was done purely in the XY plane, yet the rings that we were picking up were at a different Z plane than the top of the goal spin-

dle (this can be seen in figure 4). So not only did the actuator have to close and open, it had to move in the Z plane. The Actuator handler has two actuators, "pick_up" and "drop_off".

For each actuator, the gripper needed to move to the center of the region it was in (since the rings are precisely in the middle of each region, as the spindle is in the center of the "goal" region). I created a function _gotocenter, which took its pose, found the region it was in, found the center of said region (using previous LTLMoP code), created the offset command, and proceeded to send the gripper to the center of the region.

Both "pick_up" and "drop_off" took advantage of the "_gotocenter" function. "pick_up" goes to the center of the region, lowers to a pre-determined offset to level of the ring, and closes the gripper on the ring. "drop_off" simply navigates to the center of the region, and opens the gripper to drop the ring.

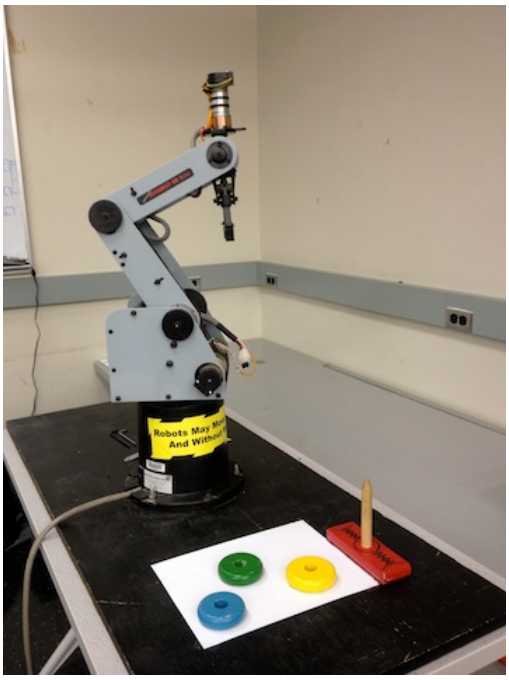

Figure 4: Finding pose from the Scorbot

4.6 Spec

The spec created was loosely based on firefighting spec. "Ring" is the dummy actuator for sensing a ring. If the Scorbot "senses" a ring in one of the regions, it will perform the "pick_up" action, which in turn activates "carrying_ring". It will then navigate to the goal, and once in the goal perform "drop_off". Once "drop_off" has been performed "carrying_ring" is then reset. So that the ring doesn't continually "sense" a ring in the goal and then perform "pick_up" (and then "drop_off" the ring again in the goal) we prevent it from sensing "Ring" in the goal.

```
#Initial conditions
Env starts with false
Robot starts with false

# Assumptions about the environment
If you were in Goal then do not Ring

# Define rules on how to pick up and drop off rings
Do pick_up if and only if you are sensing Ring and you are not activating
carrying_ring
If you are activating pick_up then stay there
carrying_ring is set on pick_up and reset on drop_off
Do drop_off if and only if you are in Goal and you are activating carrying_ring

If you did not activate carrying_ring then always not Goal

# Patrol regions
Group regions is RingRegion1, RingRegion2, RingRegion3
If you are not activating carrying_ring then visit all regions
If you are activating carrying_ring then visit Goal
```

Figure 5: Scorbot Demonstration Specification

5 Conclusion and Future Work

Operating the Scorbot-ER 5plus using LTLMoP was accomplished. Future work of having the Scorbot navigate in the Z plane using LTLMoP, while complicated would be exciting. Perhaps now with the increased use of ROS: MoveIt! more bugs will be worked out and it will be possible to utilize this tool to employ collision avoidance.