MASSACHUSETTS INSTITUTE OF TECHNOLOGY

DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS: SPACE SYSTEMS LAB

Prototyping Robotic Manipulators For SPHERES

Edward Lopez, Lisandro Jiménez, Andrew Kurtz

September 12, 2014

1 INTRODUCTION AND BACKGROUND

SPHERES is a project that is based around small satellites(8 inches in diameter) that use CO_2 thrusters and sensors to maneuver with other satellites. SPHERES are used to explore possibilities in spacecraft-to-spacecraft maneuvering and communication. SPHERES is an acronym for Synchronized Position Hold, Engage, Reorient Experimental Satellites. These satellites, aboard the International Space Station, provide a low risk experimental platform to test out different algorithms and techniques that would otherwise be difficult/expensive to test. On the ISS, these experiments can be carried out in microgravity in full six degrees of freedom. These experiments are used to explore experimental space technologies and algorithms, such as spacecraft flight formation and, in this case, robotic arms.

Having a robotic arm attached to satellites/spacecraft can be very useful, but can introduce new problems. The application of these arms can range from helping astronauts on spacewalks to autonomously constructing large structures in space. One problem is that when the arm is moved, the entire satellite moves in response.

In this project, we used LEGO MindStorms, as it provides an easy-to-work-with solution to our need to iterate rapidly. Currently, there is not a testbed for testing algorithms for spacecraft with robotic arms, and this project aimed to create one. For this project, a team of three UROPs[1] worked to develop and refine the robotic arm, develop the hardware and software involved in communicating between the arm and SPHERES, and to develop simulations of the SPHERES-arm interaction.

2 Literary Research

At the start of this project, we looked at what had been done before so that we could build off of that and create a project that would contribute to the field of space robotics. The information we compounded is located in the appendix of this report, which contains a matrix of what has and has not been done as well as a list of academic papers pertaining to the subject matter. The essence of what we learned from this was that there is quite a bit of theoretical research done but little experimentation has occurred. With that, we figured it would be best if we made an arm that it could move in 3D space and be able to simulate the system (SPHERES, Halo and the Arm) dynamics. We also thought the final, end-all be-all goal would include using Vertigo[2] to track a moving object, then using the arm to grab it, and finally using thrusters to bring to system to a stop. We never got to this though.

3 Simulation and Planning

Our first step was to simulate the dynamics that would be experienced when the arm was manipulated, and to be able to know what positions to put each joint of the arm in order to be able to get the end-effector in a desired position.

[1] Undergraduate Research Opportunity Program
[2] Other SSL project involving cameras for vision tracking. See Vertigo Homepage for more info

3.1 Inverse Kinematics

Inverse Kinematics is used to determine how a structure moves and rotates to reach a desired point. For the arm specifically, we needed to know how much to rotate each motor to reach a desired location. In 2D, the most efficient way of doing this was with circles. With a two-joint arm, all that is needed to do is draw one circle with a radius of the length of the first arm segment centered at where the arm begins and draw another circle with a radius of the length of the second arm segment centered at the desired location. Where these circles intersect is the location that the second joint (beginning of the second arm segment) needs to be in order to reach the desired location.

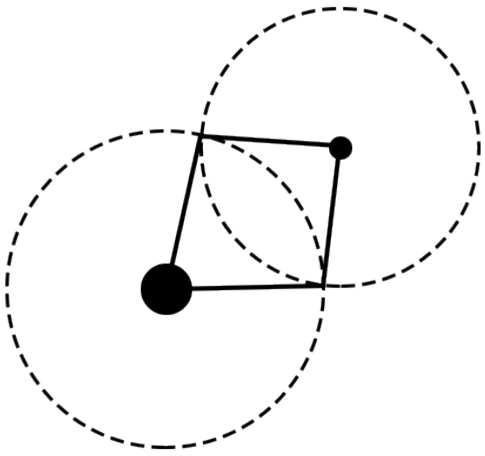


Figure 3.1: Circle Method for inverse kinematics

The same method can be used for greater degrees of freedom(more arm segments). There will be infinite solutions, however, as opposed to two, due to the nature of two dimensional space. Figure 3.2, showing one possible solution, illustrates this concept. A limited amount of circles are shown in the figure for simplicity, but in actuality there are an infinite amount of circles around the lower large circle.

To implement the Circle Method in software, we used Matlab. To do this, we used *fsolve*, a system of equations solver. The two equations were the two equations of the circles, which held their radius and center point. The variables we solved for were x_1 and y_1, the location of the second joint. From there, we did simple trigonometry to solve for the angle the first arm made with respect to the x-axis and the angle the second arm made with respect to the first

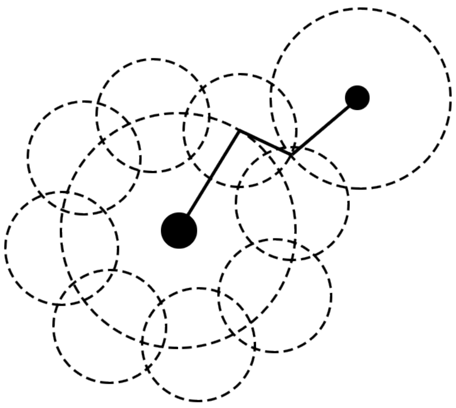


Figure 3.2: Circle Method for 3 DOF

segment. This can be expanded in two ways: 3D and more arm segments.

For more arm segments, we just need to have more circles, one for every segment. The thinking here is that you the first segment draws out one circle. The start of the second segment has to be located some point on that circle, so it is assigned an arbitrary point (x_2, y_2) on the first circle. Centered on that point, it draws out a circle with a radius of its length. The equation of that circle will look like $(x_2 - x_1)^2 + (y_2 - y_1)^2 = L_2^2$, with (x_2, y_2) representing a point on the rim of the circle. The third segment has to begin somewhere on the second circle, so it is given an arbitrary point (x_2, y_2) and draws a circle centered at that position. It's equation looks like $(x_3 - x_2)^2 + (y_3 - y_2)^2 = L_3^2$. This cycle continues until the $n'th$ segment centered at (x_{n-1}, y_{n-1}) draws a circle of its own. We want the desired location to be on that circle, so it's equation is $(x_n, x_{n-1})^2 + (y_n, y_{n-1})^2 = L_n^2$, with (x_n, y_n) representing the desired location. From there, *fsolve* did the rest of the work.

With 3D, it gets a bit more complicated. The motors used only had one degree of rotational freedom, which greatly limited them. If they had three degrees of rotational freedom, it would be simple: we would use spheres instead of circles in the method described above. But alas, life is not that easy. This is because in our design all of the arm segments lie on the same plane, which would not be considered when using spheres. To solve this problem, we thought that rotating the Cartesian axes about the y-axis would work the best. We needed to ensure that the desired point lied on the plane made by the arm. To check this, we just need to ensure $(P - D) \cdot N = 0$ is satisfied, where P is a point on the plane of the arm, D is the desired location, and N is the vector normal to the plane of the arm.

We decided to start the arm off on the xy-plane, we chose $(1,1,0)$ to be our point on the plane. Since the arm will be rotating around the y-axis, the plane will be too, so we simply applied the rotation matrix to it. We arrived that the point on the plane is described as $cos\theta\hat{\imath}+\hat{\jmath}-sin\theta\hat{k}$.

$$\begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow cos\theta\hat{\imath}+\hat{\jmath}-sin\theta\hat{k} \tag{3.1}$$

From there, we needed to describe the normal vector to the plane that the arm is in. Since we originally placed the arm xy-plane, we initially chose the normal to start at the origin and go to $(0,0,1)$. Since the plane is rotating, the normal vector will be rotating as well, so we simply applied the rotation matrix to it. After some calculations, we learned that we could describe the normal vector as $sin\theta\hat{\imath}+cos\theta\hat{k}$.

$$\begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow sin\theta\hat{\imath}+cos\theta\hat{k} \tag{3.2}$$

With that, we had all the things we needed to describe the system. Our desired point is at some arbitrary location $X_D\hat{\imath}+Y_D\hat{\jmath}+Z_D\hat{k}$. Our point on the plane of the arm is described as $cos\theta\hat{\imath}+\hat{\jmath}-sin\theta\hat{k}$, and the normal to the plane is $sin\theta\hat{\imath}+cos\theta\hat{k}$. Having those three, we can finally solve $(P-D)\cdot N=0$ for some value θ.

$$\begin{aligned} 0 &= (P-D)\cdot N \\ &= ((cos\theta\hat{\imath}+\hat{\jmath}-sin\theta\hat{k})-(X_D\hat{\imath}+Y_D\hat{\jmath}+Z_D\hat{k}))\cdot(sin\theta\hat{\imath}+cos\theta\hat{k}) \\ &= sin\theta(cos\theta-X_D)-cos\theta(sin\theta+Z_D) \\ &= sin\theta cos\theta-sin\theta X_D-cos\theta sin\theta-cos\theta Z_D \\ &= sin\theta X_D+cos\theta Z_D \end{aligned} \tag{3.3}$$

Solving for θ, we get the following result:

$$
\begin{aligned}
-sin\theta X_D &= cos\theta Z_D \\
-\frac{sin\theta}{cos\theta} &= \frac{Z_D}{X_D} \\
tan\theta &= -\frac{Z_D}{X_D} \\
tan^{-1}(-\frac{Z_D}{X_D}) &= \theta
\end{aligned}
\tag{3.4}
$$

That result tells us how much we need to rotate the first motor so that the desired location is in the plane of the arm. From there, we simply treat the problem as if there was only planar motion, using the circle method described previously.

3.2 Dynamics Simulation and Visualization

In order to predict the dynamics of the system, we came up with a formula resulting from Newton's 2nd Law. This formula describes how a two-segment system will move given the moments of inertia of both segments about the pivot point and at least one displacement angle is known.

$$\frac{\theta_1}{I_2} = \frac{\theta_2}{I_1} \tag{3.5}$$

Where I_1 and I_2 are the moments of inertia of each of the respective segments, and θ_1 and θ_2 are the respective displacement angles.

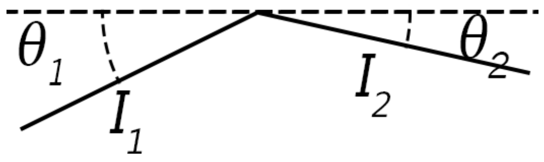


Figure 3.3: Angle Displacement Method

Another property of space manipulators is that the center of mass of the system does not change. To also satisfy this condition, one must calculate the center of mass before and after the angular displacement of each of the arms, then subtract the two to calculate the "pseudo-movement vector," which is used after the angular displacement is calculated to deduce the new correct position of the arm system.

We created a graphical simulation in C++ using the SFML graphics library in order to visualize the method we came up with to predict arm dynamics. We created a class to represent an arm-SPHERES system. This class in turn contained three "arm segment" class objects(one for the SPHERE, two for each joint of the arm). It then became easy to manipulate the joints of the arm and find out information of the system. Because of this, we were able to implement a brute force search method to find the motor positions that would result in the arm reaching a certain spot. This is independent of graphics, which allows us to run it on, for example, Halo. Using SFML, we took this information and drew basic shapes to represent each of the arm segments and the spheres. Since SFML was an external library, it was important, for portability, to make sure the core would be independent of the graphics engine.

3.3 Heading on level 2 (subsection)

4 Communication

Communication was a central topic in this project. We needed the Halo side to talk to the arm in order to provide commands for the arm to move. We also needed the arm to talk back to the Halo hardware in order to provide feedback data. Additionally, this all had to be coordinated correctly, due to the limitations of RS485. RS485 is only capable of one way communication, which complicated things, since we wanted two-way communication.

The Lego NXT is only capable of RS485 communication, which is what we used, along with an RS485 to RS232 converter and an RS232 to USB adapter. On the NXT side, we used a program written in RobotC to send and receive data. The program is made up of two modes, a listening mode and a sending mode, in order to work around RS485's limitation. During the listening mode, the NXT waits for 14 bytes worth of character data. The 14 bytes are formatted as `"xxxx,xxxx,xxxx"`, where the x's are digits. For example, a valid command could be `"100 ,45  ,5   "`. Note the whitespace after the characters; it is necessary to complete the 14 bytes. Once it has received 14 bytes, it switches into sending mode for about four seconds, where it sends state data of the position of the arms in a similar format that was used to write to it. The only difference is that the whitespace is not included. For example, a return data command could be `"-5,300,45"`. After about four seconds, it sends a terminating "!"

character, then switches back into listening mode. This whole time, there is a parallel subprocess that, using a proportional controller, controls the voltage to each of the motors to get the joints to the desired angles. The "desired angles" are updated every time 14 bytes are received by the NXT.

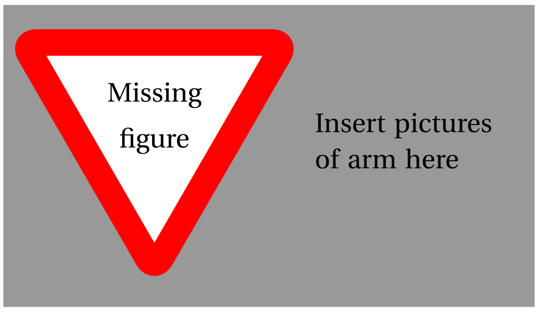
On the Halo side, we had a test project running where it would also switch between a sending and receiving mode. It would start by sending a 14 byte command, then listening until it received a "!" character. It would parse the incoming data byte by byte, and it would write a line to a log file that had the angle state of the three motors. After the "!", it would then send the next 14 byte command, and the cycle would repeat.

5 Arm Hardware

5.1 Arm Structure

We had to design a mechanical interface that would allow us to mount LEGO parts onto the Halo. We designed it so that it could accommodate a PCB and all the wires that would be needed to interface the two devices while not restricting access to any of the ports on the NXT. We also designed it to have many holes that would allow future developers to attach the LEGO components in any way they may see fit. The final piece of hardware was 3D printed in ABS plastic at a cost of about $45.

The second part of the hardware solution is the arm itself. Before arriving at the final design we explored many options, including geared drive trains, linkages, and various degrees of freedom. It eventually came down to a choice of a planar arm with all the degrees of freedom residing in the same plane. This allowed for an arm with two degrees of freedom and a claw which could be used to grab things, or an arm with 3 degrees of freedom where one degree allows rotation of the other two to a separate plane and a magnet as an end effector.



5.2 Circuitry

As described before, we had to use an RS485 to RS232 converter, then an RS232 to USB adapter. This was less than ideal, so we decided to create a PCB that would provide the same functionality, but less cluttered and more professional. We never got it fabricated, but the schematic and PCB layout are displayed below.

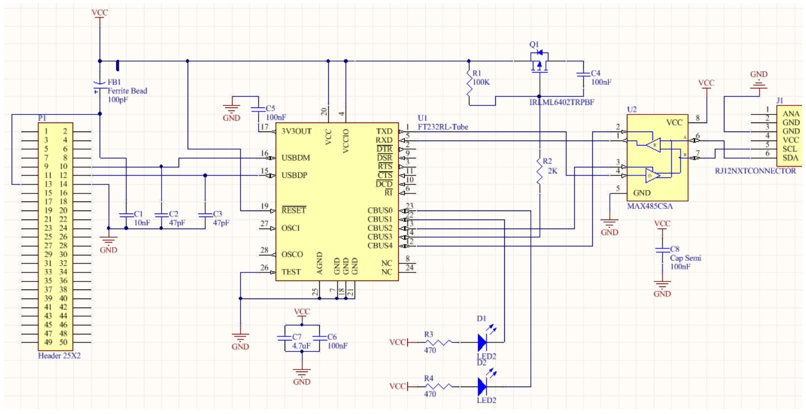


Figure 5.1: PCB Schematic

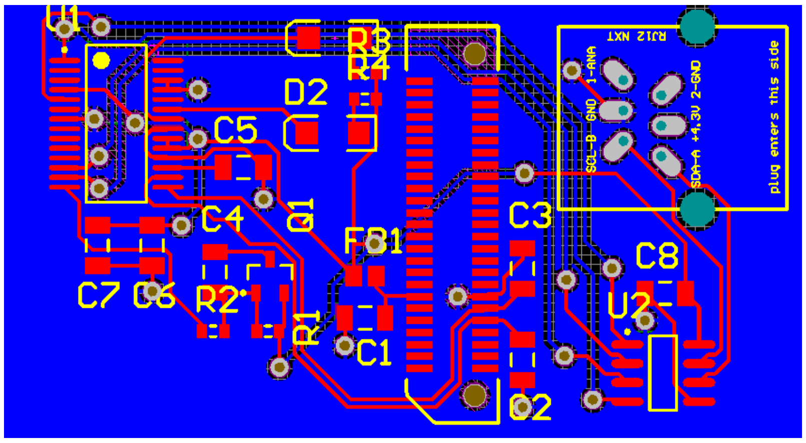


Figure 5.2: PCB Layout

6 Testing

After getting the arm functional and communicating with a Linux box, we were able to run tests. These consisted of six positions that repeated themselves over and over again. These commands were sent about every four seconds, and the arm would report back values for those four seconds, at an interval of about .105 seconds, or 38 steps every four seconds.

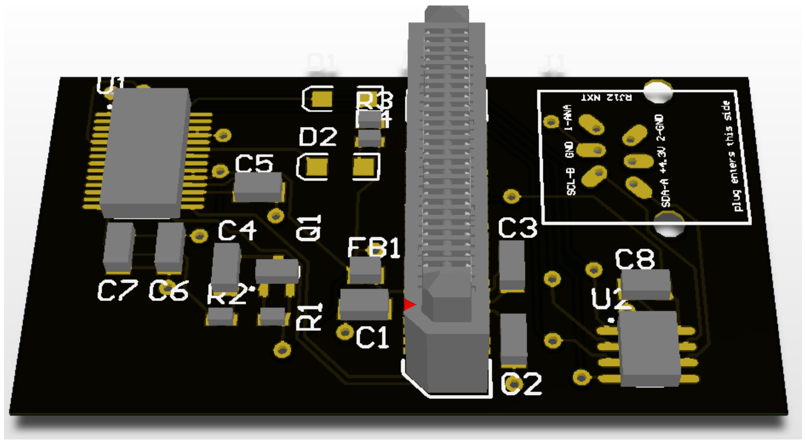


Figure 5.3: PCB Render

6.1 Glass Table Demo

We tested our arm hooked up to SPHERES and Halo, on the glass table. The day before the demo, we ran the demo on the virtual machine that simulated Vertigo, and it worked just fine. The arm moved around in the pattern that we programmed. When we hooked up the Arm to the Halo and SPHERES and ran the code on Vertigo, the Arm did not move. We noticed that the buffer size was not at 0, which as described earlier, is not good because that means the data was being sent incorrectly. After some observation and debugging, it was discovered that we needed to reset the baud rate on Vertigo to 115200, plug in the Arm after Vertigo was turned on so that we could communicate with it using ttyUSB1, and using a bunch of settings from the SPHERES to establish stable communication.

Once we solved the communication issue, we ran the demo. Overall, the test went really well, and the video of it can be found on the SSL YouTube page. There were no issues once we got the program running, but there were some things that we did not account for. When making the simulation, we did not consider the Halo and the air carriage, both of which affected how the system as a whole rotated when the Arm moved by increasing its mass and changing its moment of inertia. Also, the table seemed uneven, evident in how the contraption slide around the table, even after it was stopped. This did not change the results though.

6.2 Results Analysis

While we were unable to retrieve the motor positions as read by the arm, we had data from our tests on our Linux VM that weren't conducted on the glass table. This data is displayed below.

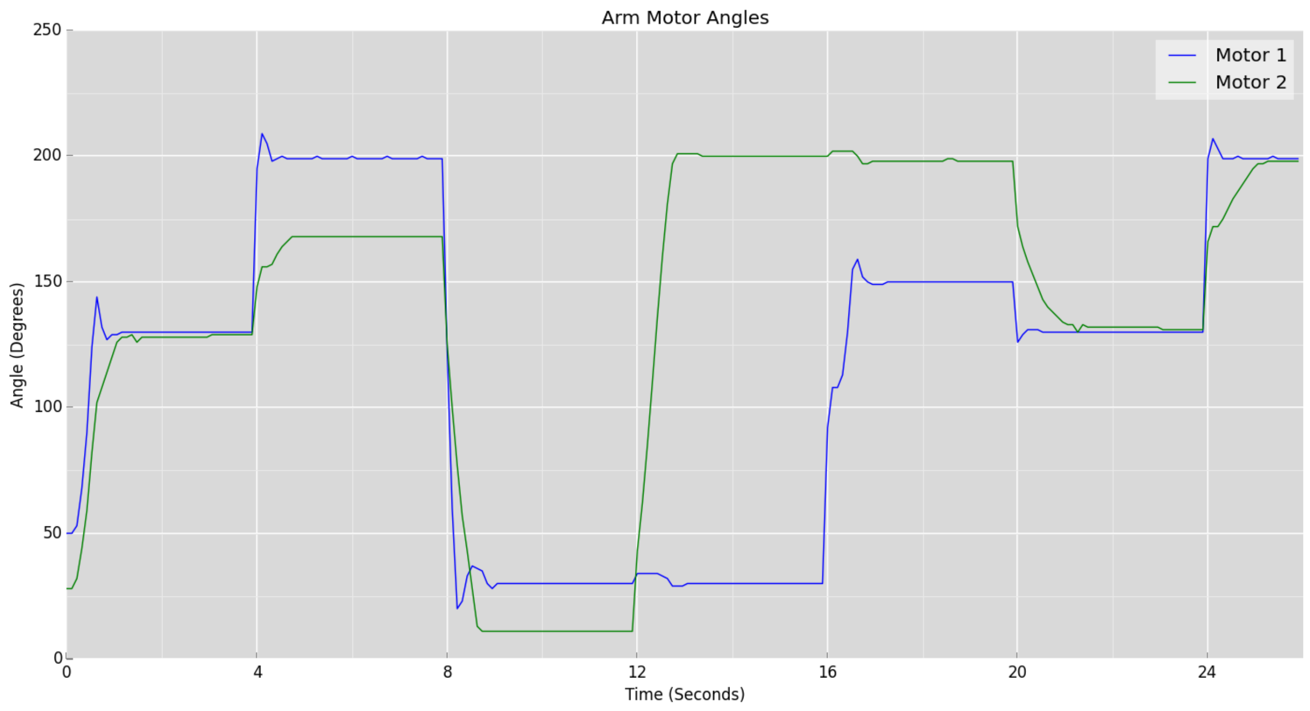


Figure 6.1: Motor Plot

An interesting feature is the slight overshoot seen by motor 1, but not motor 2. This may be because each motor is moving arm segments with different moments of inertia, but we used the same proportional controller constant for both. Changing the constant for each individual motor may result in little or no overshoot. Another interesting note is the slight movement in the non-moving motor when only one motor is moving. This is most evident at the 12-second and 16-second mark. This interaction between the two motors can also be seen even when they are both active, for example, at the 4-second mark. This probably due to the rapid acceleration of the motors moving the entire system very fast, thus resulting in a net torque in the other motor.

6.3 Future

The behavior of the arm system in the glass table test was similar to the behavior of the simulation as seen in the video. This shows that the Angle Displacement Method described in section 3.2 could be used to accurately simulate and predict the dynamics of space manipulators. The next steps for this are to accurately measure moments of inertia and masses of each of the components in the arm system. This would allow accurate prediction of the position of every part of the arm, which could then be fed into the estimator as another input source. Additionally, this could be used with the brute force search method described in section 3.2 for effective path planning. Further study of the data could also possibly lead to path planning while also using thrusters to minimize the disturbance caused by rapid movement of the arm.

7 APPENDIX

7.1 THE MATRIX

Topics	Solved Problems	Unsolved Problems
Position	Using encoders in the arm	
Movement	If we know moment of inertia and mass distribution, we can predict how the system moves. Extensively researched within one plane	Not tested in space; not tested outside of planar
Dynamics After Grabbing	Theoretically solved; Use force sensors to find mass/inertia properties of object	Not tested in space; not tested outside of planar
Oscillations	Damping oscillations caused by impacts at the end of the arm	
Catching Stuff	Robots are able to catch stuff on Earth	Catching something in space; Responding after (or before) catching to reduce rotation/movement caused by catching
Verification		Possible Solutions: Touch sensors in hand; Camera at end of the arm

Key Ideas and Possibilities:

- Implement 3rd Degree Of Freedom, since there is not a lot of research on it
- Grabbing a moving object, possibly using Vertigo/Goggles to gather information
- Return to a stationary position after grabbing an object

7.2 Literary Search

The Kinematics, Dynamics, and Control of Free-Flying and Free-Floating Space Robotic Systems by *Steven Dubowsky* and *Evangelos Papadopoulos* (1993)

- Dynamics and control
- "Three examples of promising methods for planning and controlling the motion of space robotic systems are presented."

Free-Flying Robots In Space: An Overview of Dynamics Modeling, Planning and Control by *S. Ali A. Moosavian* and *Evangelos Papadopoulos* (2007)

- Control
- Trajectory Planning
- Experimental Study
- Multiple Arms
- "Basic issues of kinematics and dynamics modeling of such systems, trajectory planning and control strategies, cooperation of multiple arm space free-flying robots, and finally, experimental studies and technological aspects of such systems with their specific limitations are discussed."

Methodology for Control of a Space Robot With Flexible Links by *K. Senda* and *Y. Murotsu* (2000)

- Path Tracking
- "This paper proposes a methodology for designing stable manipulation-variable feedback controls of flexible manipulators for positioning control to a static target and continuous path tracking control."

Coordinated Manipulator/Spacecraft Motion Control for Space Robotic Systems by *Evangelos Papadopoulos* and *Steven Dubowsky* (1991)

- Control

- "This paper studies the coordinated control of space manipulators and their spacecraft. The dynamics of free-flying space robotic systems are written compactly as functions of the system barycentric vectors. A control technique is developed that includes requirements on a statecraft's position and orientation as well as on its manipulator."

Path Planning for Space Manipulators to Minimize Spacecraft Attitude Disturbances by *Steven Dubowsky* and *Miguel A.Torres* (1991)

- Efficient Path Planning
- "A technique called the Enhanced Disturbance Map to plan space manipulator motions that will result in relatively low spacecraft disturbances is presented."

Planning of Safe Kinematic Trajectories for Free Flying Robots Approaching an Uncontrolled Spinning Satellite by *Stephen Jacobsen, Christopher Lee, Chi Zhu,* and *Steven Dubowsky* (2002)

- Catching Objects
- "The problem of planning a safe trajectory for a free-flying robot to approach an uncontrolled spinning satellite is addressed."

Maneuvering and Vibrations Control of a Free-Floating Space Robot with Flexible Arms by *Ramin Masoudi* and *Mojtaba Mahzoon* (2011)

- Motion
- Vibration Suppression
- "A free-floating space robot with four linkages, two flexible arms and a rigid end-effector that are mounted on a rigid spacecraft; is studied in this paper."

Dynamic Simulations of Free-Floating Space Robots (Chapter 30) by *Tomasz Rybus, Karol Seweryn, Marek Banaszkiewicz, Krystyna Macioszek, Bernd MÃď diger,* and *Josef Sommer* (2011)

- Physical Testing
- Dynamics
- "This paper focuses on the dynamics of a 6-dof manipulator mounted on a free-flying servicer satellite during final part of an on-orbit rendezvous maneuver."