

ENSC 488/894

OpenGL Simulation of a Haptic Robotic Device for Monitoring the Elderly

Lee Sutton 301145106

Contribution: 33.33%

Nasreen Mohsin 301283129

Contribution: 33.33%

Maikel Saad 200126606

Contribution: 33.33%

1.0 Introduction

With the average global age gradually increasing, the need for elderly assistance is needed now more than ever. One possible solution to aid the aging population would be to use robots that can interact with the elderly in some form. Some robotic applications might include monitoring the elderly in case of a fall, cooking for the elderly who may have lost the ability, or assisting with daily medical care. The purpose of this project is to simulate a robotic environment that could be used to monitor an elderly person in their home. The simulation uses OpenGL linked with C++ to create a graphics enabled program that includes an interactive haptic robotic device. The program allows the user to move the robot to any point in the environment, and orient the camera to the desired viewing angle.

1.1 Open GL

Open GL was developed in January of 1992 to provide an open source graphics application programming interface. OpenGL is short for open graphics library. It can be linked with nearly all programming languages to create a graphics enabled library. It is used to interact with a graphics processing unit to create 2D and 3D vector graphics. Since 1992 it has undergone many revisions and rebuilds. OpenGL V4.5 was used in this project linked with C++.

2.0 Modelling and Design

2.1 Robotic Arm

The robotic arm is the most critical aspect of the project, it allows the camera to orient itself within the workspace. A large workspace and degrees of freedom are both desirable. However, when adding degrees of freedom the kinematics of the arm become much more complicated. A balance is thus needed between the two features, complexity and degrees of freedom. Four possible designs were evaluated for their possible use in the project, an RRP, an RRR, an RRPR, and a PUMA 560 design. To limit the scope of this project, the PUMA 560 robotic arm design was chosen as the inverse kinematic solutions and the forward kinematic solutions have been derived in the past. For an arbitrary design, these solutions can become extremely complicated or not exist, by utilizing the PUMA 560, our group could focus on building the simulation environment and implementing the known kinematic solutions. The PUMA 560 also offers a large workspace and enough freedom to orient the camera in any way.

2.2 Mobile Platform

The mobile platform was designed to allow the robot to move to locations not in its current workspace. The mobile platform must be able to turn the robot 360 degrees and move in at least one direction. This would allow the robot to reach any position in the floor space. Typically a mobile platform would consist of three or four wheels, two of which are driven with differential drive to allow for rotation (FIGURE). Our design uses a novel approach to allow for easier maneuvering. Our design consists of 4 caster wheels and one driven wheel. The driven wheel is oriented using a servo motor and then driven by a DC motor. This allows for the robot to be rotated 360 degrees and also driven in any direction (FIGURE). This would be very useful in application for obstacle avoidance (FIGURE 2).

2.3 Hardware Specifications

Please note the figures referred to in this section can be found in Appendix A. Hardware was specified based on the requirements given for this project. DC motors were specified to be used at the joints and to drive the base. The device is intended for monitoring the elderly in their home as a result, a camera

was specified for the end effector. For controlling the motors, a motor controller was specified in accordance with the DC motors being used. In addition, a micro controller was specified to control the system operation.

2.3.1 Motor Specifications

We chose a dc motor due to the fact that it has the best qualities compared to other motor options as can be clearly seen in figure 7. The 57BL60L2 motor, by Molon motors and coil corp, was chosen based on its characteristics and dimensions comparison with other options in its class. Its advantages include low input power required, high output current and torque. It is capable of an output speed of 4000 rpm and a torque output of 0.145 Nm. the input voltage is 24 VDC. Its power rate is 60Watts. The resistance and inductance of the motor are 0.3 Ohm, 0.32 mH respectively as displayed in figure 2.

Figure 1a displays a simplified version of a brushless DC motor; the motor of choice. A brushless dc motor is capable of fast response of the rotor to the drive current in the stator. For maximum torque and a much smoother rotor acceleration, control of the stator polarity is used. Stator windings can be excited by a 1-D current or by a bi-directional current, which is termed pole switching as displayed in figure 1c. When applying pole switching in the stator windings using a sinusoidal wave, high torque (Fig: 1c), fast response and smooth movement of the rotor can be maximized.

discrepancies between the drive, or desired, motion and the actual output parameters of the motor can be minimized using a sensor in a feedback loop which can detect the output parameters of the motor as displayed in figure 1d. The sensor of choice is the hall sensor, which is the most widely used in industry for the brushless DC motor.

A brushless DC motor is the choice due to the many disadvantages of a dc motor with brushes. A brushed DC motor has many disadvantages including rapid wear out, mechanical loading, wear and heating due to sliding friction, contact bounce, excessive noise, and electrical sparks, and voltage ripples at switching points. brushless motor advantages include high efficiency, low mass, low maintenance, longer life, and quieter operation.

2.3.2 Camera Specifications

The camera of choice is the Sharp GP2Y0A710K0F. It uses infrared sensor to keep track of objects. The range is 1-5.5 meters, which is acceptable for our applications.

2.3.3 Joints

The joints will be constructed from a strong plastic through the process termed water jet cutting. The plastic of choice is HDPE due to its characteristics. The mass of each joint is calculated by determining the length, width, and thickness of the joint, calculating volume and multiplying by the density.

2.3.4 Wheels

The PUMA haptic base design of choice is to support the base by 4 wheels not driven by motors and a fifth wheel in the center of the base which is driven by 2 motors. For the fifth wheel 1 motor is connected along the axis of the wheel for linear (back/forward) displacement and the second motor is connected above the wheel for rotational motion of the robot as displayed in figure 8. The rectangle on the left of the wheel is used as a mass to counterbalance the weight of the motor on the right of the wheel. This is needed in order to eliminate any undesired inertia caused by the mass of the motor during rotation. The wheels of choice for all wheels is the long-lasting nylon wheels. These wheels are

made of extra hard 65D-85D durometer material, never need lubrication, have maximum service life, and same capacities as steel. For the center wheel with the motion control drive is chosen to be

2.3.5 Motor Controller

The PUMA haptic design of choice requires 6 motors for the joints as well as 2 more motors for motion of the wheels. The HBL2360 is capable of controlling 8 motors as well as driving the motors with 3 phases and sinusoidal waves as well as enabling the motors to provide high torque through current direction switching.

2.3.6 Micro-Controller

The microcontroller of choice is Arduino Mega 2560 Microcontroller as it is suitable for our design. The control of the PUMA is as follows.

2.3.7 Control Logic

The microcontroller intakes the information about the objects in the environment from the infrared camera then computes the direction of motion in order to avoid collision. Motion of the joints is translated by the microcontroller from the user interface then the desired joint motion is sent to the motor controller, which controls the motors driving the joints. Smooth motion, no lag time, high speeds, as well as high torque is achieved by the three-phase control by the motor controller.

2.4 Final Device

The final device is shown below in FIGURE. It was constructed in Solid works and scaled according to the DH table shown above. The simulated device was also constructed in OpenGL this is shown below in Figure 1.



Figure 1 CAD model of the final device

3.0 Methodology

3.1 Kinematics and DH - Parameters

The selected device was to be simulated in an OpenGL environment. The program was designed to allow the user to move the robot throughout the environment and orient the camera to monitor any position. Movement was designed to be controlled in two ways via the forward kinematics or the inverse kinematics. Details on the forward kinematics and the inverse kinematics can be found in sections 3.3 and 3.4 respectively. However, before deriving the kinematic solutions, frame attachments needed to be assigned, and the DH parameters needed to be recorded. The frame attachments are shown below in FIGURE and the corresponding DH parameters are shown below in table 1. The DH parameters and the frames were assigned to mimic the PUMA 560 model. This allowed us to implement the derived forward kinematic solutions and inverse kinematic solutions in our simulation.

Table 1: DH Parameters

Link Frame i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	D1	θ_1
2	-90°	0	0	θ_2
3	0	a2	0	θ_3
4	-90°	0	d4	θ_4
5	90°	0	0	θ_5
6	-90°	0	0	θ_6

3.2 Static Tipping Point

To calculate the tipping point of the robot a simplified model was constructed. The torque required to tip the robot is a function of the direction the torque is applied. For a conservative estimate we examined the direction which would create the smallest tipping torque. The direction that would cause smallest tipping torque is shown below in Figure 2. When the robot begins to tip the normal force acting on the back wheel will approach zero. Using the mass of the robot base we found that torque required to tip the robot was $T = 35\text{Nm}$.

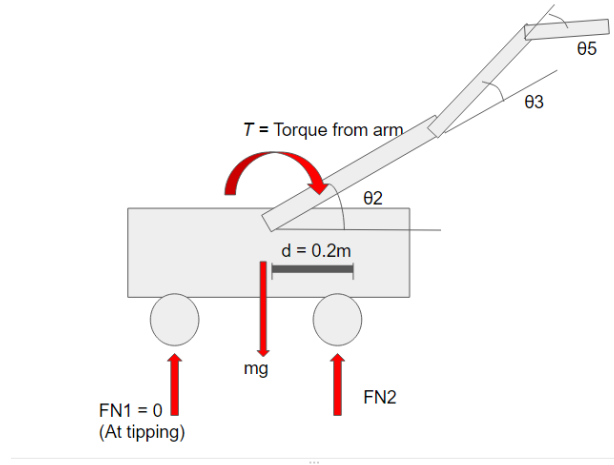


Figure 2 Free body diagram of the robot base

The torque created by the weight of the camera and the motors was a function of all the joint variables. However, to simplify this calculation on $\theta2$, $\theta3$, and $\theta4$ were used in the calculation (details can be found in Appendix D). Using this methodology in combination with the minimum tipping point estimated above, this would provide a conservative estimate of whether or not the robot will tip. At the tipping point the normal force to the left is zero and so is the torque associated with it. However torque from the normal force $N2$ is calculated as follows: $T = N2 \cdot d$. $N2$ is calculated after the base mass is determined. T is determined to be 35Nm. The maximum torque that the armature can exert on the base is when all angle values are zero. However, the tipping point is determined using the code in appendix D. the tipping torque is the torque applied by the armature which is equal to the torque applied by $N2$. The tipping armature torque is determined by the minimum joint angles which provide the tipping torque value equal to T . each joint torque includes the joint mass inertia applied at the center of the joint, the motor mass inertia applied at the full length of the joint. For the last joint the camera weight is added to the motor weight. Appendix D includes all calculations as well as the code described.

3.3 Forward Kinematics

The forward kinematics take in the joint variables, $\theta1$ to $\theta6$ and compute the position and orientation of the end effector. The forward kinematics are used in this simulation to find the location and orientation of the camera based on the user input. The forward kinematics for the proposed device resembles the forward kinematics for the PUMA 560 robot. The degrees of freedom and frame attachments are identical to the PUMA 560 Figure 3. Since the forward kinematic solutions for the PUMA 560 are well known they can be applied here [1]. The equations for the forward kinematic solutions can be found in Appendix B.

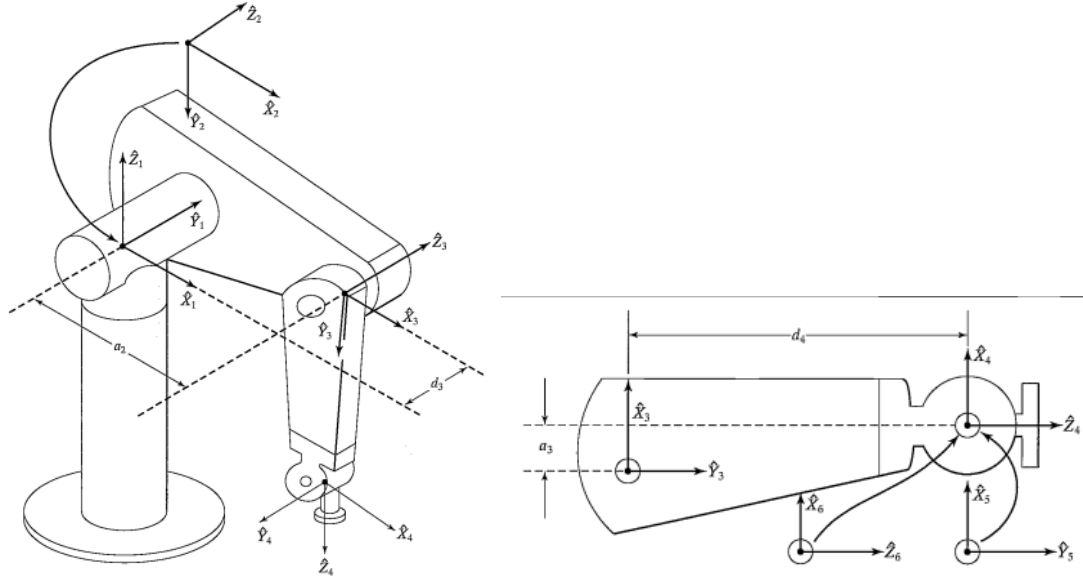


Figure 3 Frame attachments for Puma 560

3.4 Inverse Kinematics

The inverse kinematics take in the desired position and orientation of the end effector and calculate the joint angles, theta 1 to theta 6 to get to the desired position. The inverse kinematics allow the user to input the desired transformation matrix and the camera will orient itself in this way. The inverse kinematics for the proposed device resembles the forward kinematics for the PUMA 560 robot. Since the inverse kinematic solutions for the PUMA 560 are well known they can be applied here [1]. The derivation of the inverse kinematic solutions can be found the Appendix.

3.5 Trajectory Planning

The robot joints and links are required to move to from their initial position to final position in smooth manner. Initial position and orientation are known in the form of a set of joint angles θ_i . In addition, final positions and orientations are calculated from the inverse kinematics thereby giving out another set of angles θ_f . In order to make a single smooth function, cubic polynomial of the form is used as shown below.

$$\theta(t) = a_0 + a_1t + a_2t^2 + a_3t^3,$$

In order to make velocity continuous in the derivative of the cubic polynomial, the four constraints were used

$$\theta(0) = \theta_0,$$

$$\theta(t_f) = \theta_f.$$

$$\dot{\theta}(0) = 0,$$

$$\dot{\theta}(t_f) = 0.$$

With desired final time t_f set and the four constraints, the coefficients of the cubic polynomial can be calculated as shown below[1].

$$\begin{aligned}a_0 &= \theta_0, \\a_1 &= 0, \\a_2 &= \frac{3}{t_f^2}(\theta_f - \theta_0), \\a_3 &= -\frac{2}{t_f^3}(\theta_f - \theta_0).\end{aligned}$$

These coefficients are used to generate desired trajectory for the monitoring robot thereby connecting initial joint angles to final joint angles.

4.0 Software Architecture

The program consists of 4 main components, allowing the user to update the forward kinematics, trajectory planning, allowing the user to orientation matrix to be used with the inverse kinematics and checking for tipping. Details for each component are discussed in this section.

4.1 Forward Kinematics

4.1.1 Robotic Arm

The forward kinematics involves rotating the arms of the robot based on the given joint variables. In other words, the joint variables are specified by the user and the program rotates the arms by these angles. To implement this in the simulation, the rotate functions were utilized in conjunction with hierarchical modeling. Hierarchical modeling is the process of applying multiple transformation in a chronological order. This concept is illustrated below in Figure 4 on a simplified robotic arm. First the arm is constructed using an OpenGL function, and scaled to the proper dimensions. After this the PushMatrix would be cleared so that the translation and rotation could happen. The arm is then be translated so its edge aligned with the axis of rotation and then allowed to rotate. The next arm was constructed in a similar way using a built in function in OpenGL. The arm was then translated twice so it would align with the end of the previous arm. It would then be allowed to rotate about its axis of rotation. This process is illustrated below in Figure 5 and Figure 6.

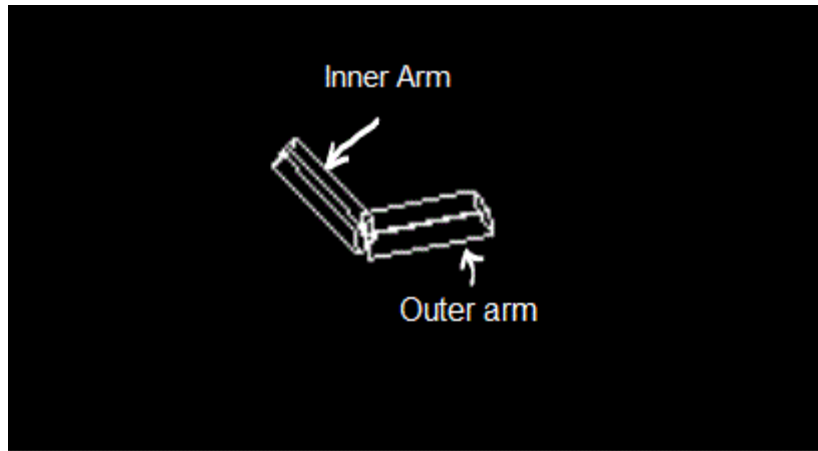


Figure 4 Simplified robotic arm to demonstrate the forward kinematics methodology

For the inner arm

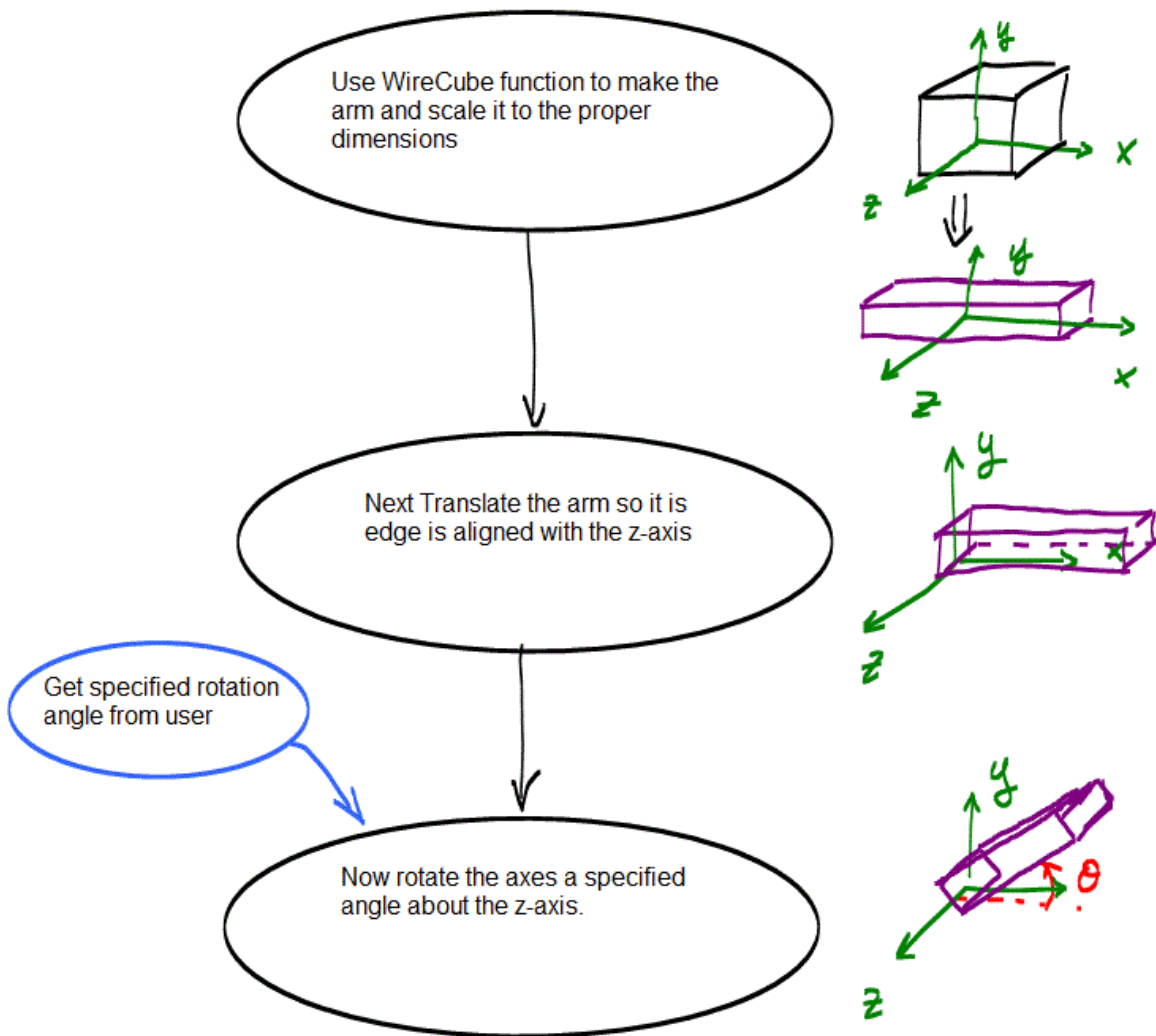


Figure 5 inner arm rotation for forward kinematics

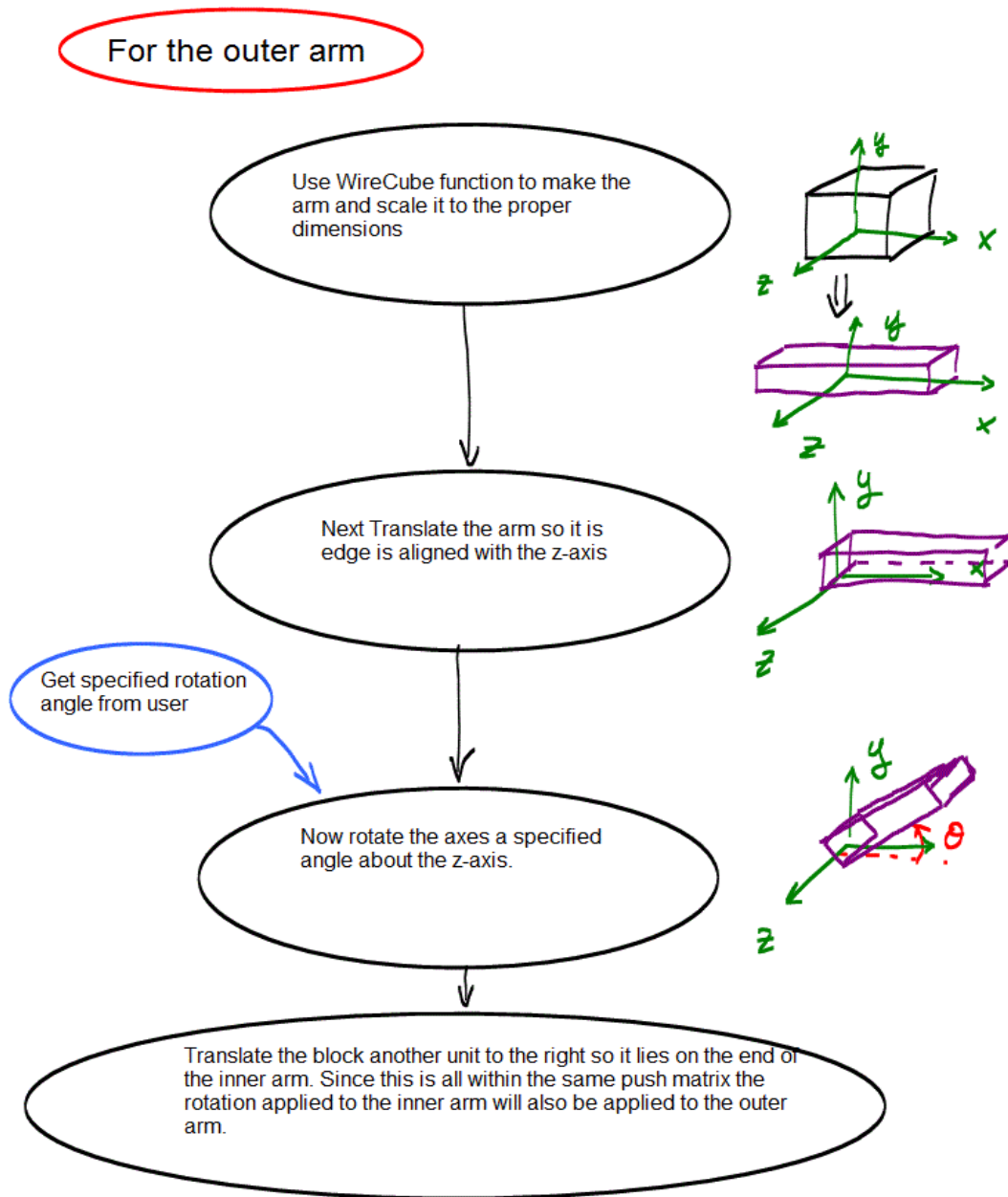


Figure 6 Outer arm rotation to demonstrate the forward kinematics method

The concept shown above was easily extended to include multiple arms and different axes of rotation. However, it was important to note that all of the rotation axes had to be made with respect to the world coordinates. Each reference frame in the phantom haptic device was constructed as an “arm” as above. In the real haptic device there is only three physical arms however, the above method only allows the arms to rotate around a single axis. Therefore, each joint on the haptic device was constructed as an arm in OpenGL. If the coordinate frames were supposed to align, like frame {4}, {5}, and {6}, the arms were made very small and they all intersected at the same point. They were then assigned a rotation axes with respect to the {0} frame in OpenGL..

4.1.2 Mobile Platform

The mobile platform was designed to provide easy maneuvering in the given environment. Details can be found above in section 2.2 this platform was constructed in the OpenGL simulation and the forward kinematics were used to move the device throughout the floor space. Based on the wheel design shown above in section 2.2, the simulation was designed to allow the Puma to maneuver in any direction. When given a desired point, the base will move to that point in the workspace. It used the `glTranslatef` function with the trajectory function to create a smooth motion. Based on the wheel design shown above in section 2.2, the simulation was designed to allow the Puma to maneuver in any direction.

4.2 Inverse Kinematics Architecture

The inverse kinematics is used to orient the camera to the given position. The program takes in input from the user, checks if the point is within the workspace and orients moves itself to that point. If the point is outside the workspace, the robot will move its base so that the desired point is within the workspace. The joint angles will then be calculated using the inverse kinematic equations and moved to the desired location. Details are shown below in Figure 7.

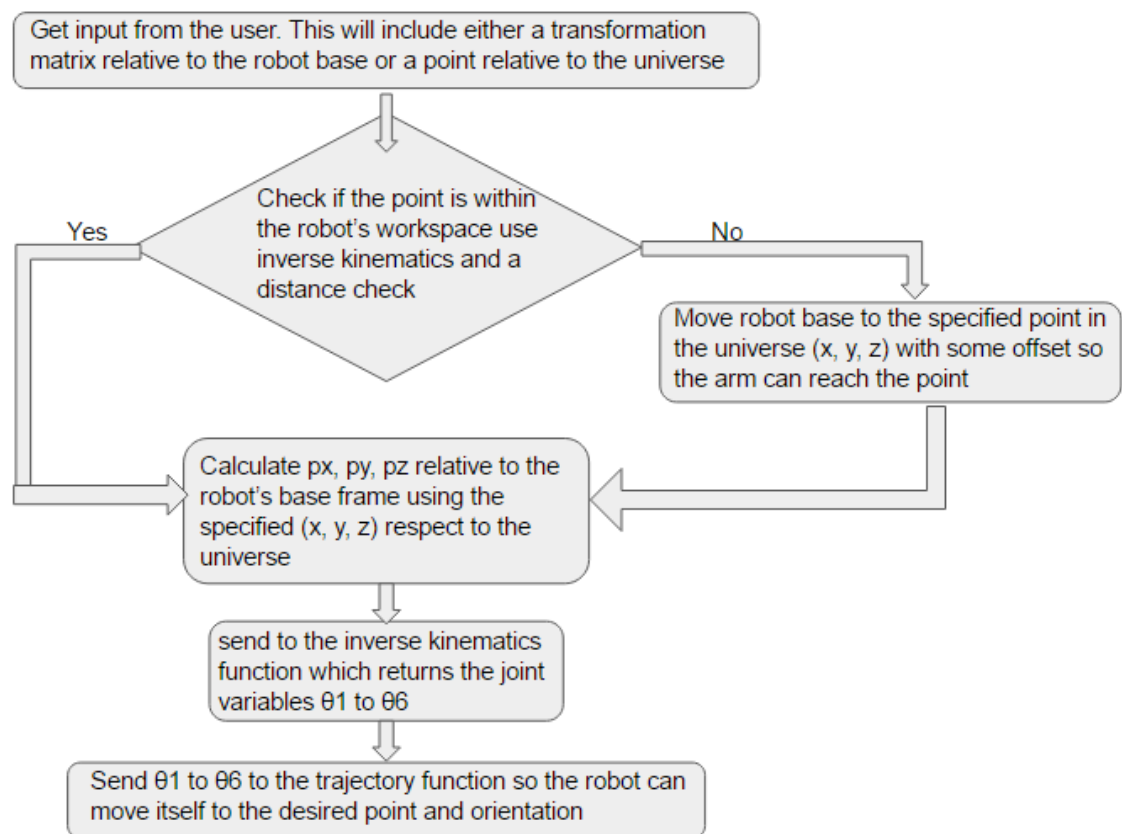


Figure 7 Inverse kinematics flow chart

4.2.1 Inverse Kinematic Function

The inverse kinematics function was built to take in a transformation matrix and calculate the corresponding joint angles. As shown in Appendix C, the inverse kinematics formulas lead to 8 valid

solutions for the joint variables. As a result, all of the theta values needed to be checked to ensure they fall within the joint limits. Once a valid solution was found that falls within the joint limits, this solution is returned. If no valid solutions can be found, a variable is returned indicating the point is out of the workspace. If this was the case the robot base was moved until the point fell within the workspace. It is important to note that this function did not necessarily find the best solution, it only returned the first valid solution. Possible methods to find the best solution are noted in the project limitations section.

4.3 Trajectory Planning

After obtaining the newly calculated angles of the robot's joints, these angles are then plugged into the trajectory mapping function along with the initial set of joint angles and desired time duration. The trajectory mapping function was formulated based on the section 3.5. At the each given point of time, the function gives out the new set of joint angles which in turn send to forward kinematics function. The forward kinematics function then render the motion of the robot links in the simulation window.

5.0 Simulation Results

The simulation included the inverse and forward kinematics. The code is included in the directory. Snapshots are shown below of some of the simulation results. The first figure shows the device when the program is initialized. The program also shows when the robot might tip by changing the base colour to red. Images are also shown of the device moving to its destination.

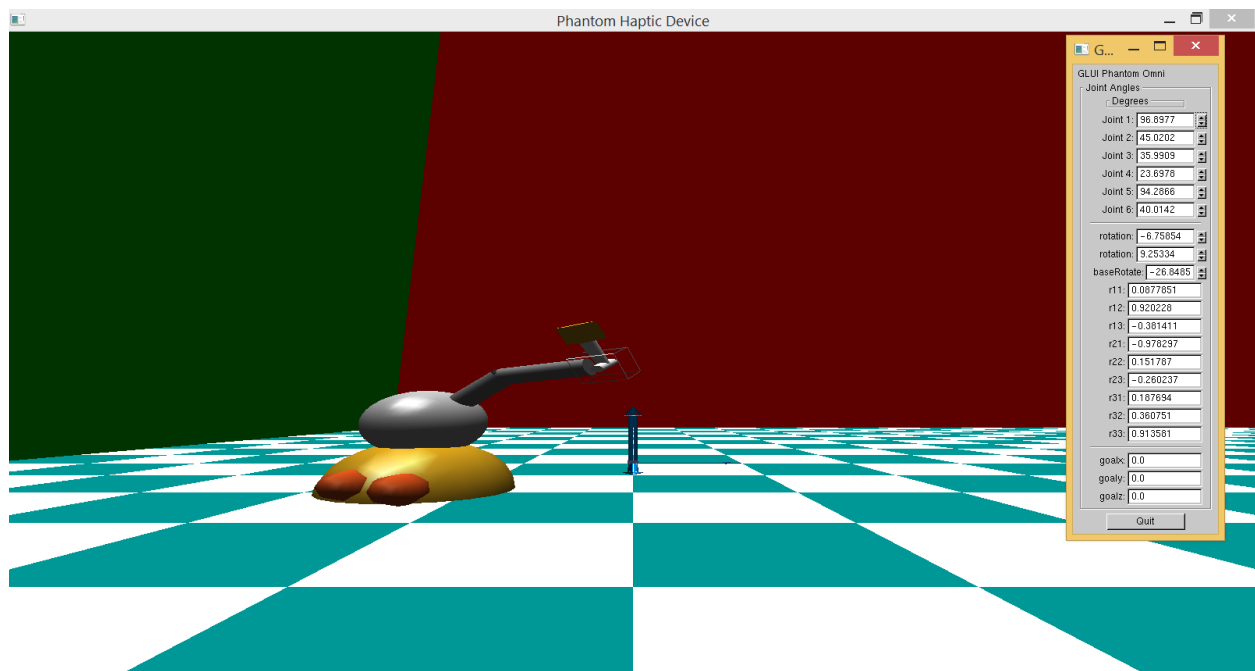


Figure 8 Program initialization

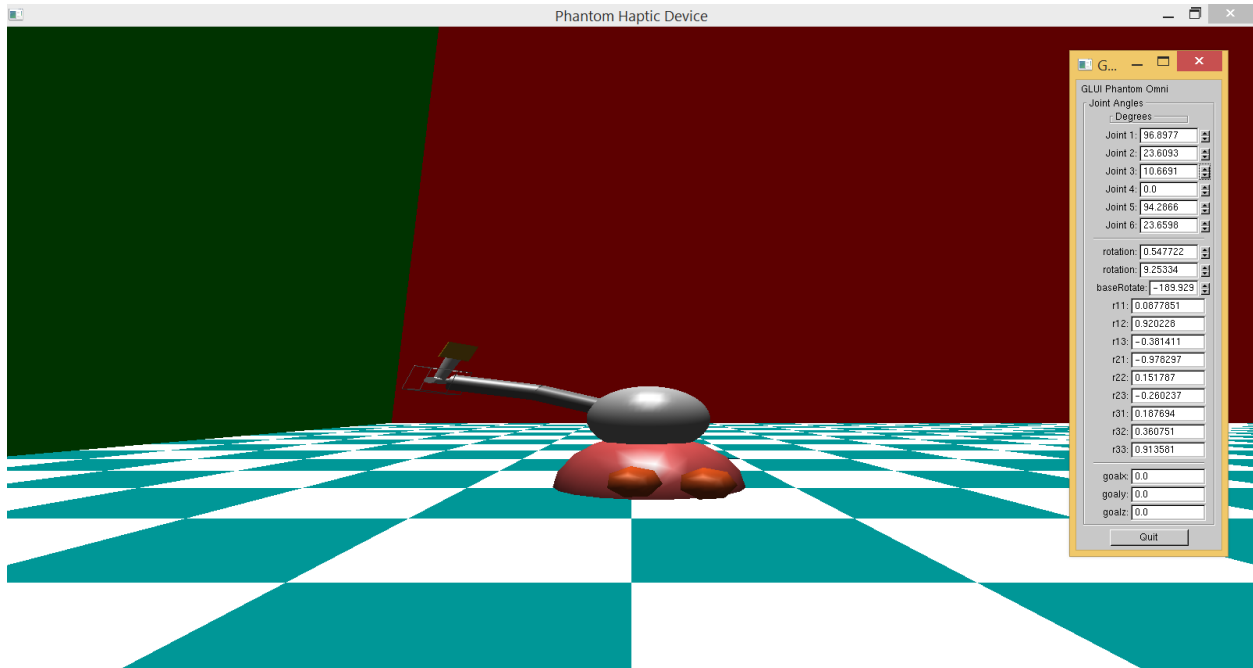


Figure 9 Base changing to red when tipping

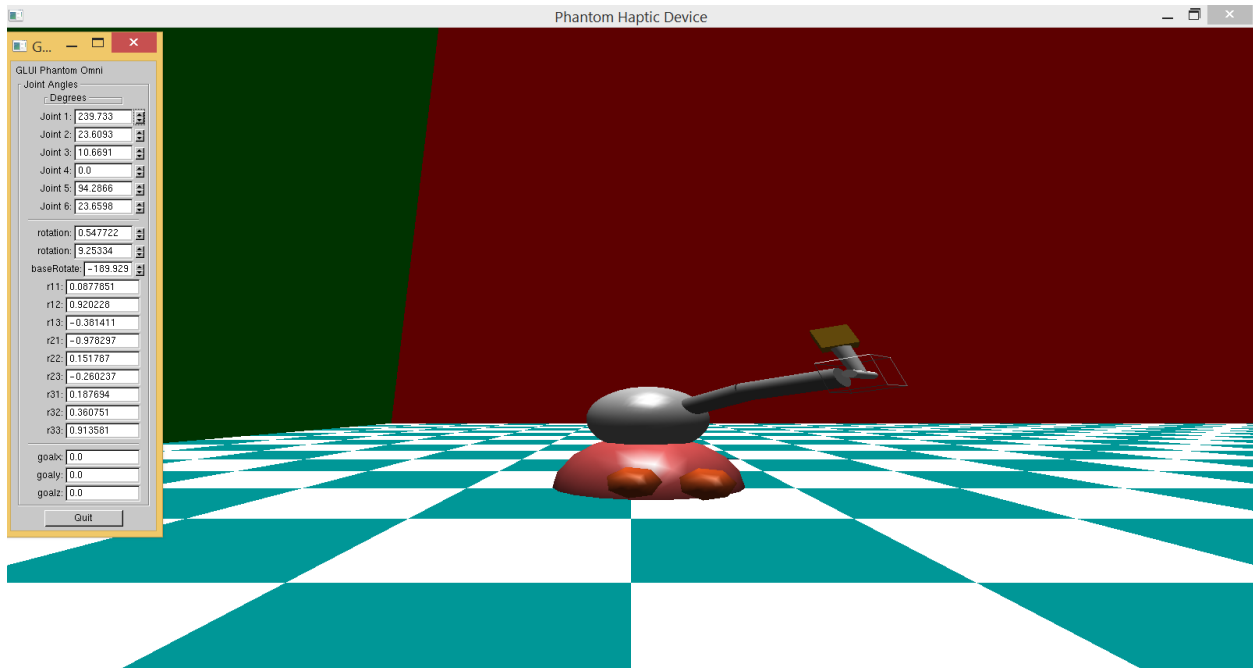


Figure 10 Another orientation which could cause the base to tip

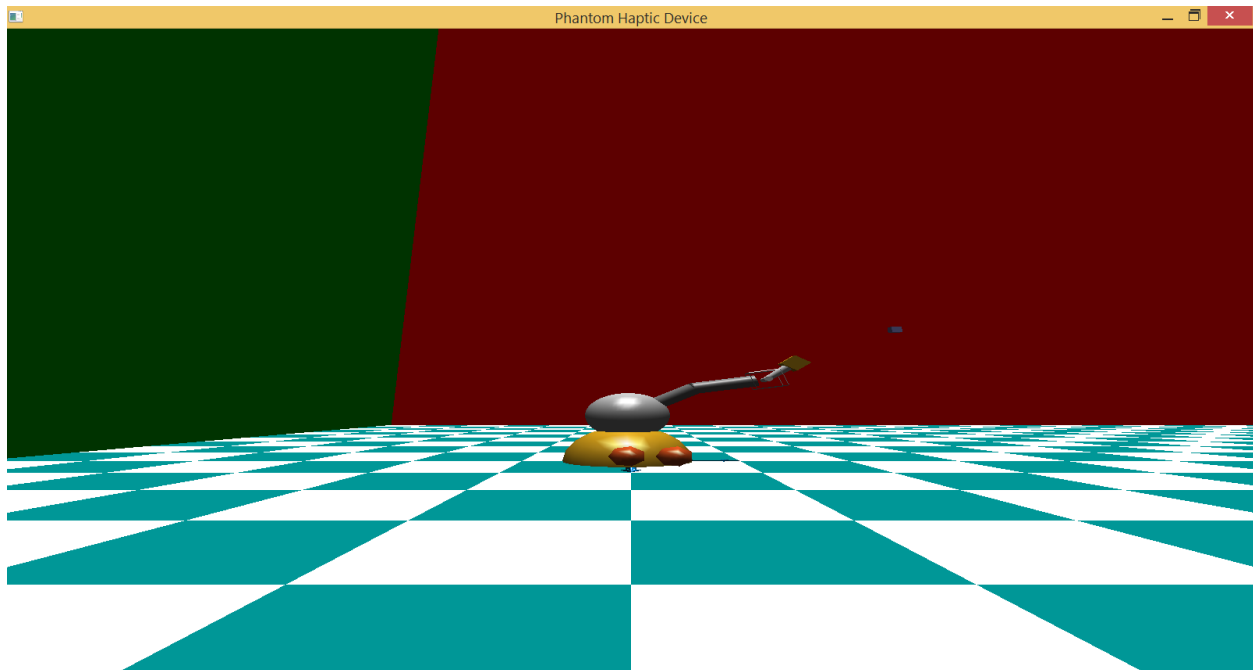


Figure 11 Device before moving to its goal destination

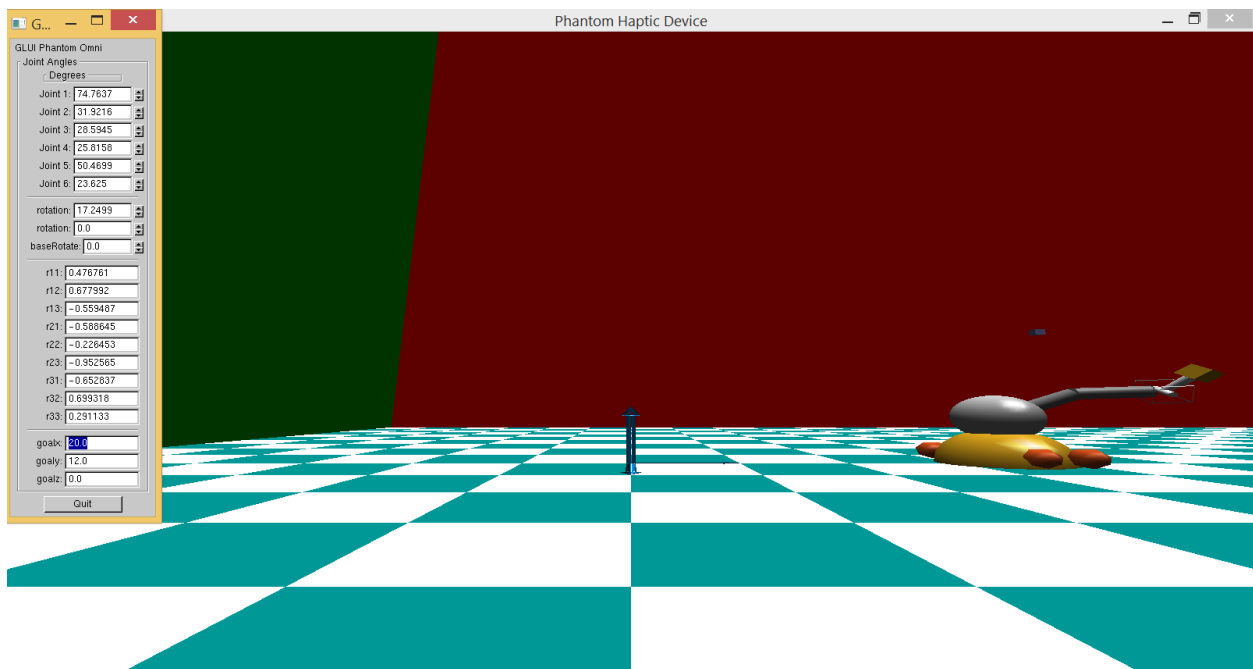


Figure 12 Device moving to its input orientation

6.0 Project limitations

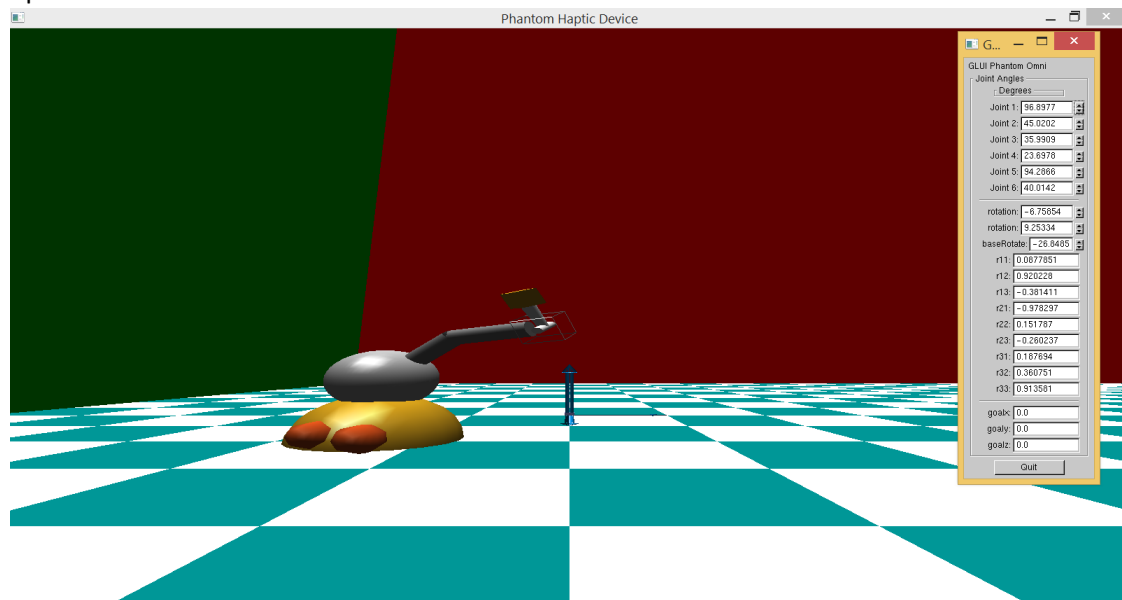
After creating the simulation, a number of bugs were found in the program. These bugs are listed below.

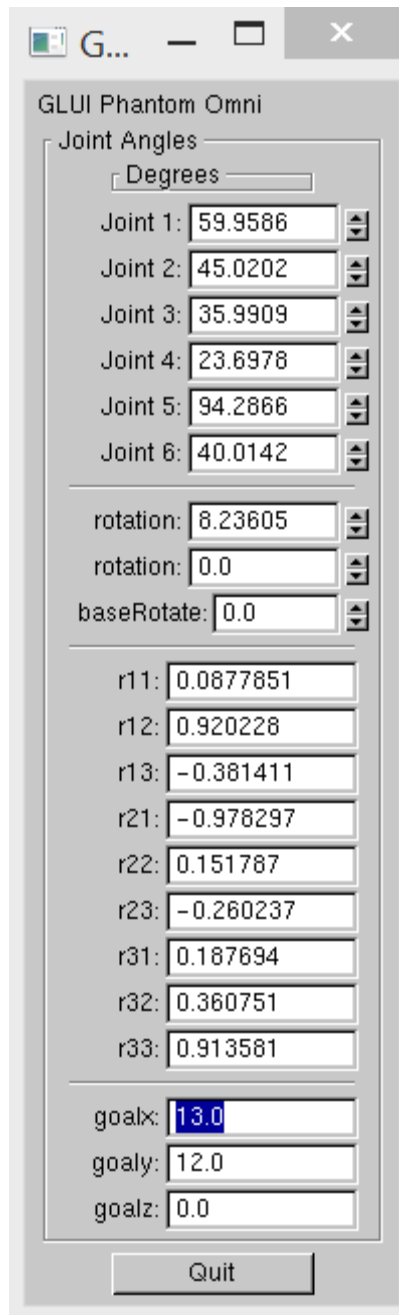
- If the theta values change too fast, the system might not be able to render the image on the screen. As a result, the arm does not appear on the screen until the motion is finished. The arm then re-appears on the screen at the final position.
- There are no restrictions preventing the device from going through the floor. Since this is possible with all the theta values within the joint limits, this can occur in the program given certain inputs. Future work could be done on this project to implement collision dynamics.
- If the input transformation is within the workspace (assuming 360 degrees for all of the joint limits) but outside of the joint limits, the arm will move to the given location. However, the final position is outside the joint limits so it will revert back to a point within the joint angles, closest to the given point.
- If the joint angles are updated using the forward kinematics, the transformation matrix in the GUI will not update. The transformation matrix in the GUI can only be changed by manually typing the values into the GUI.
- The inverse kinematics return the first valid solution that is calculated, it does not necessarily calculate the best solution. This could be implemented given more time. The function could take in the current theta values and compare them with all eight solutions from the inverse kinematics, the solution with the shortest path (and within the joint limits) could be used.

Given more time, these limitations could be dealt with. Future work could improve on this work by fixing the known bugs, creating a more detailed environment, and developing an algorithm in the inverse kinematics to return the best solution.

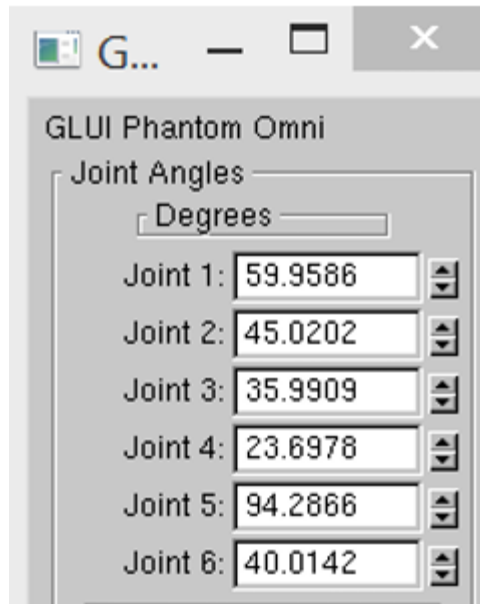
7.0 User Manual

- 1) As soon as the GroupProject.cpp is compiled and running, the following the window will be open.

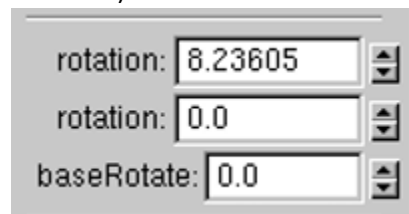




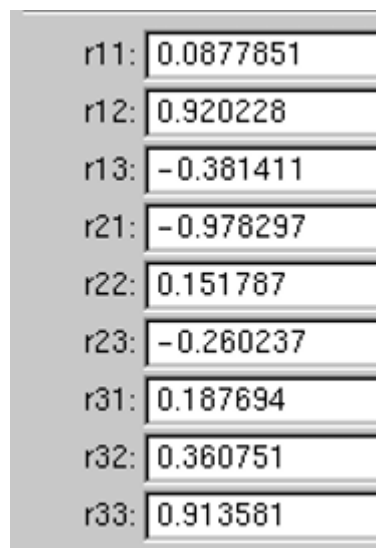
- 2) In the GUI display window, you can vary values of Joint 1-6 of the robot. Each Joint angles has its own given angle limits.



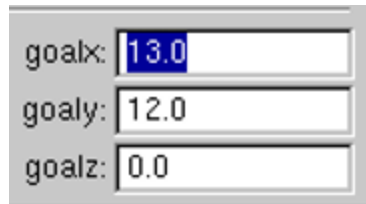
- 3) The wheels of the robot can be rotated in a direction in the GUI box rotation 1 (for x direction) and rotation 2 (for z-direction). BaseRotate box rotates the base of the robot.



- 4) Orientation of the camera with respect to the base of robot can be varied in these boxes shown below.



- 5) The co-ordinates of the goal frame can be specified in this box.



goalx: 13.0

goaly: 12.0

goalz: 0.0

- 6) You can click the button Quit to close the simulation window.



goalx: 13.0

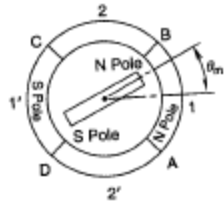
goaly: 12.0

goalz: 0.0

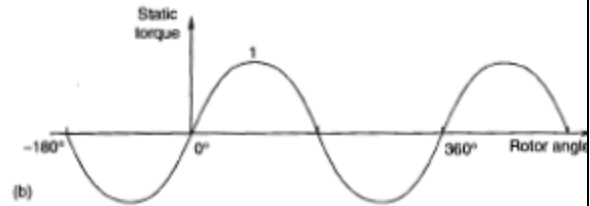
Quit

APPENDIX A

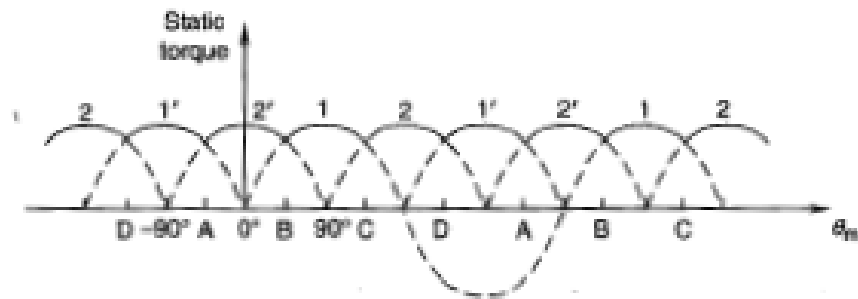
Figure 1: BRUSHLESS DC MOTOR ANALYSIS



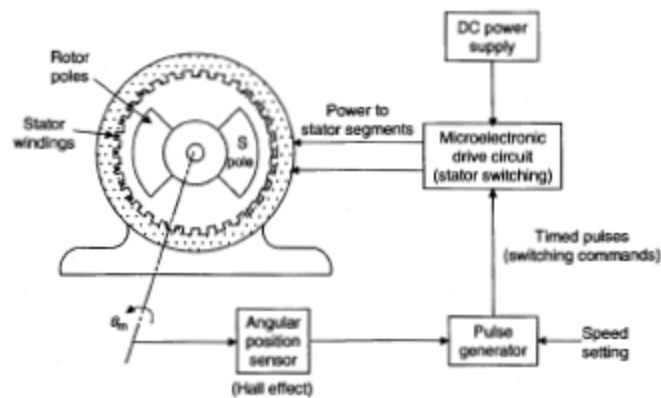
a) Brushless DC motor



b) stator current poles constant



c) torque maximization by applying alternating current direction in each stator winding



d) feedback control logic

Technical drawing of a 16-channel relay module, showing side and top views with dimensions and a pin list.

Side View Dimensions:

- Total width: 31 ± 1
- Mounting hole diameter: $\varnothing 8 \pm 0.005$
- Mounting hole offset: 7 ± 0.1
- Distance from mounting hole to center: 24
- Distance from mounting hole to edge: 2.5
- Distance from center to edge: 8
- Distance from center to edge: 1.1 ± 1
- Distance from center to edge: 5
- Total height: 34.0 ± 0.2

Top View Dimensions:

- Overall square footprint: $\square 57 \pm 0.1$
- Inner square footprint: $2 \cdot 47.14 \pm 0.1$
- Central circular area diameter: $\varnothing 4.3 \pm 0.1$
- Distance from center to edge: $\varnothing 38.10 h7$

Pin List:

r	$\varnothing 0.080$	A
b	0.100	B

Pin Numbers: 1, 2, 3, 4, 5, 6, 7, 8

[illegible]

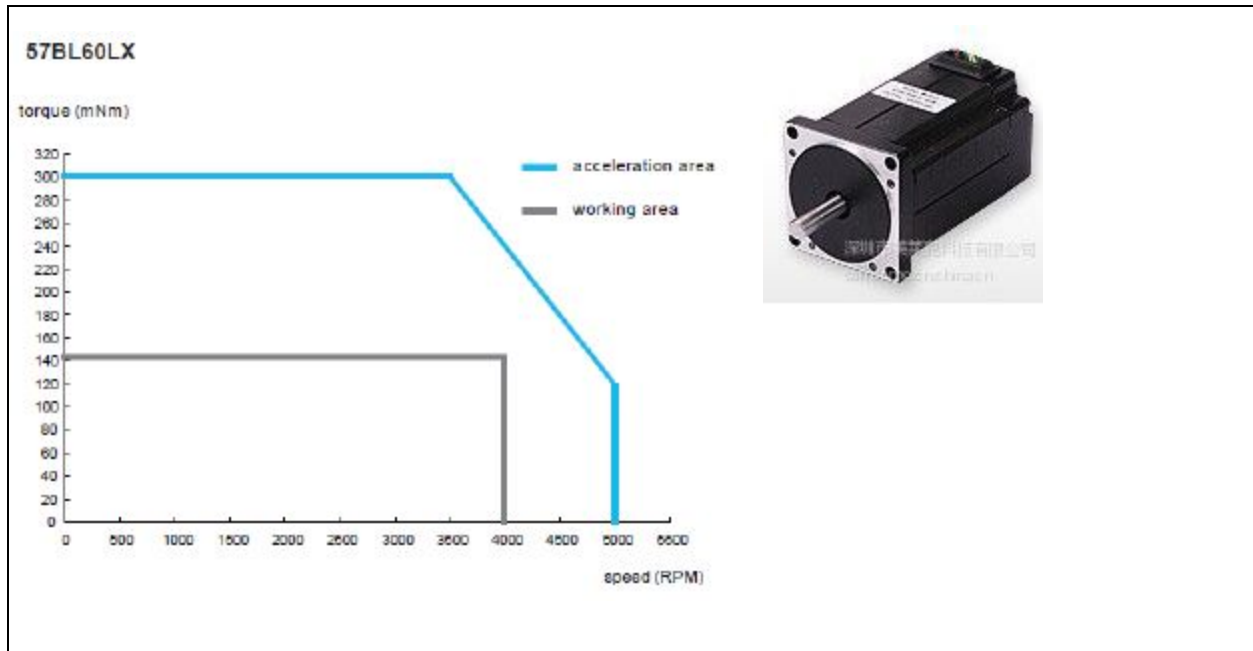


Figure 3: INFRARED CAMERA IMAGE & SPECIFICATIONS

Measuring distance range Min 100 to Max 550 cm

- Designed to use in variety of Applications areas
- Operating supply voltage 4.5 to 5.5 V
- Operating temperature -10 to +60 C
- Large format: 58.0×17.6×22.5 mm

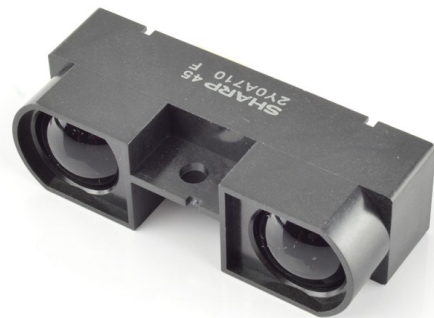


Figure 4: brushless DC motor controller



Power

- Motor type: Brushless DC
- Max voltage: 60V DC
- Number of channels: 2
- Direction: Forward/Reverse
- Max amps per channel: 75A
- Continuous amps per channel: 50A
- On resistance (mohm): 6 mohm
- Power connections: wires

Commands & feedback

- Analog: Yes
- RS232: Yes
- USB: Yes
- CANbus: Yes

Input/Output

- Max analog inputs: 11
- Max digital inputs: 19
- Digital outputs: 8
- Max pulse inputs: 6
- Encoder: Yes

Mechanical

- Cooling: Heatsink extrusion
- Operating environment: -40° to +85°C

- | | |
|--|--|
| <ul style="list-style-type: none">• RC pulse: Yes• Microbasic scripting: Yes• Control loop (ms): 1 | |
|--|--|

•

Figure 5: MICROCONTROLLER IMAGE & SPECIFICATIONS

User friendly USB programmable Arduino
Microcontroller

- Open Source design based on the larger ATmega2560
- 54 digital I/O Pins and 16 analog I/O Pins
- 256 KB of Flash Memory, 8 KB of SRAM, and 4kB of EEPROM
- Clock Speed: 16 MHz

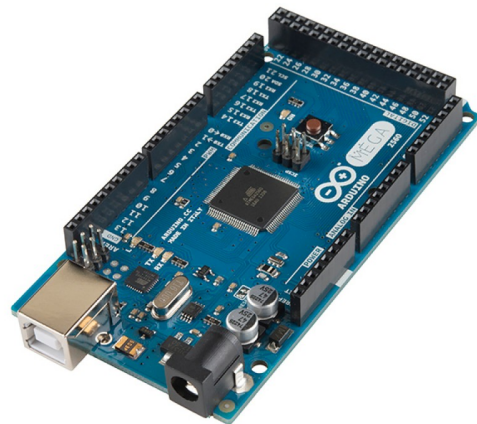


Figure 6: JOINT MATERIAL SPECIFICATIONS
HDPE IS USED

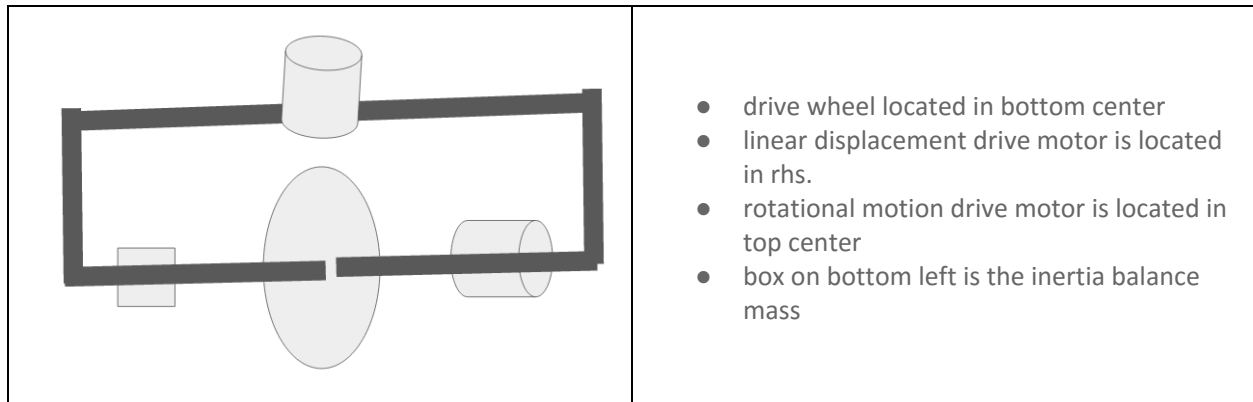
Material	Product	Tensile Strength, psi	Rockwell Hardness	Impact Strength, ft.-lbs./in.	Coefficient of Friction	Dielectric Strength, volts/0.001"	Water Absorption, %	Density, lbs./in. ³	Thermal Expansion, in./in./°F	Machine With
ABS	ABS	5,100-6,100	R102-109	5.2-7.7	Not Rated	450-1,220	0.3-1.0	0.032-0.038	5.2×10^{-5}	HSS
ABS/PVC	Electrically Conductive ABS/PVC	4,500	R87	2	Not Rated	Not Rated	Not Rated	0.04	4.6 to 5.5×10^{-5}	Carbide
Acetal	Acetal	6,400-9,500	M51-M88	1-1.8	0.11-0.35	420-500	0.2-0.8	0.048-0.051	5.4 to 12×10^{-5}	HSS
	Delrin® Acetal Resin	9,000-11,000	M89-M94	1-2.4	0.2	435-500	0.2-0.4	0.051	4.7 to 12.2×10^{-5}	HSS
	Glass-Filled Delrin® Acetal Resin	8,700	M81	0.8	Not Rated	450	Not Rated	0.054	3.33×10^{-5}	Carbide
	PTFE-Filled Delrin® Acetal Resin	6,800-12,490	M77-M78	0.7-1.2	0.07-0.14	400-500	0.25	0.054	5.1×10^{-5}	HSS
	Turcite Acetal	5,900-7,800	M63-M81	0.54-0.57	0.22-0.3	Not Rated	0.2	0.053	5.0×10^{-5}	HSS
Acrylic	Cast Acrylic	8,000-11,250	M94-M103	0.04-0.5	Not Rated	400-430	0.2-0.8	0.043	3.5 to 4.2×10^{-5}	Carbide
	Extruded Acrylic	8,100-11,030	M68-M95	0.3-0.7	Not Rated	430-760	0.2-0.4	0.043	3.0 to 4.0×10^{-5}	Carbide
Acrylic/PVC	Kydex Acrylic/PVC	6,100	R94	15	Not Rated	Not Rated	0.05	0.049	3.8×10^{-5}	HSS
Cellulose	Acetate	4,500-8,000	R78-R120	2.0-8.5	Not Rated	250-600	2.0-7.0	0.048	5.6 to 8.3×10^{-5}	Cut with knife
	Butyrate	4,800	R78	4.5	Not Rated	300-475	1.4	0.027	6.0 to 9.0×10^{-5}	HSS
CPVC	CPVC	7,100-7,300	R116-119	8-9	Not Rated	1,250	0.03	0.053-0.056	3.9×10^{-5}	Carbide
CTFE	CTFE	4,860-5,710	Shore D85-D95	2.5-3.5	0.08	500	0	0.034-0.08	3.9 to 5.1×10^{-5}	HSS
FEP	FEP	3,000	R25	No Break	0.25	1,800	<0.01	0.078	4.6 to 5.8×10^{-5}	Carbide
HDPE	HDPE Polyethylene	4,000-4,100	Shore D80-D88	1.1	0.22-0.62	450-1,800	0	0.034	5.3 to 10×10^{-5}	HSS
LDPE	LDPE Polyethylene	3,100-6,100	Shore D42-D56	Not Rated	Not Rated	Not Rated	Not Rated	0.033	Not Rated	HSS

Figure 7: MOTOR COMPARISON: BRUSHLESS DC MOTOR IS BEST FOR ROBOTIC APPLICATIONS

P4-

Characteristic	DC Motor	Torque Motor	Stepper Motor	Induction Motor	AC Synchronous Motor
Power Capability	Low-Average	Average	Low	High	High
Speed Controllability	Good	Good	Good	Average	Average
Speed Regulation	Average to Good	Average to Good	Average	Good	Excellent
Linearity	Average to Good	Average	Poor	Poor	Poor
Operating Bandwidth	Good	Good	Low	Good with Frequency Control	Good with Frequency Control
Starting Torque	Good	Good	Good	Average	Nil
Power Supply Requirements	DC	DC	DC and AC	AC	AC
Commutation Requirements	Split-Ring & Brushes	None	None	None	Slip-Ring & Brushes
Power Dissipation	Average	Average	Average to High	Low	Low

Figure 8: Wheel driving base



Appendix B – Forward Kinematics

The equations used to calculate the forward kinematic solutions are shown below.

$$r_{11} = c_1 * (c_{23} * (c_4 * c_5 * c_6 - s_4 * s_5 * s_6) - s_{23} * s_5 * s_6) + s_1 * (s_4 * c_5 * c_6 + c_4 * s_6)$$

$$r_{21} = -s_1 * (c_{23} * (c_4 * c_5 * c_6 - s_4 * s_6) - s_{23}) + c_1 * (s_4 * c_5 * c_6 + c_4 * s_6)$$

$$r_{31} = -s_{23} * (c_4 * c_5 * c_6 - s_4 * s_6) - c_{23} * s_5 * c_6$$

$$r_{12} = c_1 * (c_{23} * (-c_4 * c_5 * c_6 - s_4 * c_6) + s_{23} * s_5 * s_6) + s_1 * (c_4 * c_6 - s_4 * c_5 * s_6)$$

$$r_{22} = s_1 * (c_{23} * (-c_4 * c_5 * c_6 - s_4 * c_6) + s_{23} * s_5 * s_6) - c_1 * (c_4 * c_6 - s_4 * c_5 * s_6)$$

$$r_{32} = -s_{23} * (-c_4 * c_5 * s_6 - s_4 * c_6) + c_{23} * s_5 * s_6$$

$$r_{13} = -c_1 * (c_{23} * c_4 * s_5 + s_{23} * c_5) - s_1 * s_4 * s_5$$

$$r_{23} = -s_1 * (c_{23} * c_4 * s_5 + s_{23} * c_5) - c_1 * s_4 * s_5$$

$$r_{33} = s_{23} * c_4 * s_5 - c_{23} * c_5$$

$$p_x = c_1 * (a_2 * c_2 + a_3 * c_{23} - d_4 * s_{23}) - d_3 * s_1$$

$$p_y = s_1 * (a_2 * c_2 + a_3 * c_{23} - d_4 * s_{23}) + d_3 * c_1$$

$$p_z = -a_3 * s_{23} - a_2 * s_2 - d_4 * c_{23}$$

Appendix C

The inverse kinematic solutions are shown below. In total 8 solutions will be calculated.

$$K = \frac{Px^2 + Py^2 + Pz^2 - a2^2 - a3^2 - d3^2 - d4^2}{2 * a2};$$

$$theta1 = \left(atan2(Py, Px) - atan2(d3, sign1 * sqrt(Px^2 + Py^2 - d3^2)) \right); = cos(theta1);$$

$$s1 = sin(theta1);$$

$$theta3 = \left(atan2(a3, d4) - atan2(K, sign3 * sqrt(a3^2 + d4^2 - K^2)) \right); = cos(theta3);$$

$$s3 = sin(theta3);$$

$$t23 = atan2((-a3 - a2 * c3) * Pz - (c1 * Px + s1 * Py) * (d4 - a2 * s3), (a2 * s3 - d4) * Pz + (a3 + a2 * c3) * (c1 * Px + s1 * Py)); theta2 = (t23 - theta3);$$

$$s23 = \frac{(-a3 - a2 * c3) * Pz + (c1 * Px + s1 * Py) * (a2 * s3 - d4)}{Pz^2 + (c1 * Px + s1 * Py)^2};$$

$$c23 = \frac{(a2 * s3 - d4) * Pz + (a3 + a2 * c3) * (c1 * Px + s1 * Py)}{Pz^2 + (c1 * Px + s1 * Py)^2};$$

$$theta4 = atan2(-r13 * s1 + r23 * c1, -r13 * c1 * c23 - r23 * s1 * c23 + r33 * s23); c4 = cos(theta4);$$

$$s4 = sin(theta4);$$

$$s5 = -(r13 * (c1 * c23 * c4 + s1 * s4) + r23 * (s1 * c23 * c4 - c1 * s4) - r33 * (s23 * c4));$$

$$c5 = r13 * (-c1 * s23) + r23 * (-s1 * s23) + r33 * (-c23);$$

$$theta5 = atan2(s5, c5); s6$$

$$= -r11 * (c1 * c23 * s4 - s1 * c4) - r21 * (s1 * c23 * s4 + c1 * c4) + r31 * (s23 * s4);$$

$$c6 = r11 * ((c1 * c23 * c4 + s1 * s4) * c5 - c1 * s23 * s5) + r21 * ((s1 * c23 * c4 - c1 * s4) * c5 - s1 * s23 * s5) - r31 * (s23 * c4 * c5 + c23 * s5);$$

$$theta6 = atan2(s6, c6); theta1 = theta1 * 180/pi;$$

$$theta2 = theta2 * \frac{180}{pi};$$

$$theta3 = theta3 * \frac{180}{pi};$$

$$theta4 = theta4 * \frac{180}{pi};$$

$$theta5 = theta5 * \frac{180}{pi};$$

$$theta6 = theta6 * \frac{180}{pi};$$

Appendix D

Static Torque Calculations and Code

```
*/  
// Return a true a false value if true robot is going to tip if false it is not  
int torque(joint angles)  
{  
    //plastic density = 0.034 lbs / in ^ 3 * (25.4 / 1000) ^ 3 * 2.2 lbs / kg = 2.94488189  
    float density = 2.94488189;  
  
    //area density = density * width * thickness ....width = 0.04m and thickness = 0.00635m;  
    float d = density*0.04*0.00635;  
    float g = 9.8; //gravity = -9.8m/s^2  
  
    //m = brushless dc motor mass  
  
    float m = 0.63;  
    float mc = 1.0;  
  
    float L1 = 0.25; //meter  
    float m1 = L1 *d;  
    float t2 = angles.t2;  
    float T1 = L1*m1*g*0.5*cos(t2);  
  
    float L2 = 0.2; //meter  
    float m2 = L2 *d;  
    float t3 = angles.t3;  
    float T2 = m2*g*(L1*cos(t2) + L2*0.5*cos(t3));  
  
    float L3 = 0.1; //meter  
    float m3 = L3*d + 5;  
    float t4 = angles.t4;  
    float T3 = m3*g*(L1*cos(t2) + L2*cos(t3) + 0.5*L3*cos(t4));  
  
    t4 = angles.t4;  
  
    float TM1 = m*g*L1*cos(t2);  
    float TM2 = m*g*(L1*cos(t2) + L2*cos(t3));  
    float TM3 = (m + mc)*g*(L1*cos(t2) + L2*cos(t3) + L3*cos(t4));  
  
    //float L4 = 0.2; //meter  
    //float m4 = 0.5 + 5; // camera weight  
    //float T4 = (L2*cos(t2) + L3*cos(t3))*(m4*g);  
  
    ///cout << T4 << "\n";  
  
    float T = T1 + T2 + T3 + TM1 + TM2 + TM3;
```

```
// Set a tip variable = 0
int tip = 0;

// Tipping torque
float Torque_tip = 35;

if (T >= Torque_tip) {

    // If the torque causes the robot to tip return 1
    tip = 1;
}

return tip;

}
```